FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

CARRERA DE ESPECIALIZACIÓN EN
SISTEMAS EMBEBIDOS

MEMORIA DEL TRABAJO FINAL

Módulo de conectividad WiFi/Bluetooth para electrodomésticos

Autor:
Ing. Matías Nicolás Brignone

Director:
Esp. Ing. Diego Fernández (FIUBA)

Jurados:
Mg. Ing. Gonzalo Sánchez (FIUBA, FAA)
Esp. Ing. Matías Álvarez (FIUBA)
Esp. Ing. Santiago Germino (FIUBA)

Este trabajo fue realizado en la ciudad de Córdoba,
entre marzo de 2019 y abril de 2020.

Resumen

En la presente memoria se describe el desarrollo de un módulo que permite dotar de conectividad WiFi y Bluetooth a un electrodoméstico, a los fines de permitir su manejo remoto por parte del usuario final. También permite la recopilación de información de uso y estado por parte del fabricante del electrodoméstico. Con el objetivo de agilizar el desarrollo, el electrodoméstico a controlar se emula con un microcontrolador con el que se comunica el módulo.

Para desarrollar el trabajo se utilizaron un sistema operativo de tiempo real, diferentes protocolos de comunicación en distintas capas (WiFi/Bluetooth para la capa física, HTTPS/MQTT para la capa de aplicación, entre otros), sistemas de control de versiones y herramientas de gestión de proyectos.

Agradecimientos

A mi familia, por su constante apoyo.

A mis amigos y colegas, por acompañarme en este camino.

Índice general

Índice de figuras

Índice de Tablas

Dedicado a mi familia.

Capítulo 1

Introducción General

En este capítulo se realiza una introducción al concepto de Internet de las Cosas, se describe la motivación del trabajo realizado y se presentan sus objetivos y alcance.

1.1. Internet de las Cosas

Desde la aparición de Internet a finales del siglo pasado, ha quedado demostrado lo útil que es contar con un dispositivo capaz de conectarse a la red. Los beneficios de que una computadora o un teléfono inteligente puedan conectarse a Internet son evidentes, y esos beneficios también se encuentran presentes al conectar cualquier otro objeto a Internet, y es allí donde surge el concepto de Internet de las Cosas (IoT, por sus siglas en inglés correspondientes a *Internet of Things*).

La Internet de las Cosas consiste en extender el potencial de Internet y la conectividad más allá de las computadoras y celulares, e incorporar a todos los objetos (cosas) de la vida cotidiana y que se encuentran presentes en el entorno de una persona. Así se permite tanto la comunicación e interacción entre sí de estos objetos, como así también el monitoreo y control en forma remota.

El concepto de «cosa» es sumamente amplio, ya que contempla desde lámparas, cerraduras y termostatos en el hogar, hasta maquinaria industrial y sistemas de riego para agricultura, pasando por vehículos autónomos, sistemas de iluminación y sensores para estacionamiento público en una ciudad. El objeto conectado incluso puede ser un monitor cardíaco en el interior de una persona o un chip insertado en un animal de granja. Gracias a la existencia de sensores y microcontroladores cada vez más potentes, más pequeños y de menor costo, es posible lograr que prácticamente cualquier dispositivo forme parte del ecosistema IoT.

Los beneficios que la Internet de las Cosas ofrece tanto a empresas como a personas individuales son innumerables. Desde un punto de vista económico, les da la posibilidad a las empresas de conocer y monitorear mejor sus procesos, para así hacerlos más eficientes y posibilitar mejores tomas de decisiones y acciones como mantenimiento predictivo de maquinaria. Todo eso se traduce en definitiva en un ahorro de dinero y tiempo para la organización. Con respecto a un individuo particular, la Internet de las Cosas le permite mejorar su calidad de vida y su comodidad en el día a día con automatizaciones en el hogar (domótica), como así también beneficios a la salud con, por ejemplo, implantes inteligentes que permiten un cuidadoso monitoreo.

Las empresas, las personas y el mercado en general hace ya varios años que se han dado cuenta de la importancia de la Internet de las Cosas. Es prueba de ello la gran cantidad de dispositivos conectados que existen, y las proyecciones que indican que estos números seguirán aumentando notablemente. En 2017 ya había 27 mil millones de dispositivos IoT conectados, cifra que aumentará a un ritmo de un 12 % anual hasta llegar a 125 mil millones para 2030 [1]. Por otra parte, en la figura 1.1 se puede observar cómo el mercado mundial de la Internet de las Cosas ya maneja cantidades de dinero superiores a los USD 200.000.000.000, y cómo esa cifra aumentará sustancialmente a lo largo de los próximos años [2], lo cual supone una enorme oportunidad de negocio para muchas empresas.

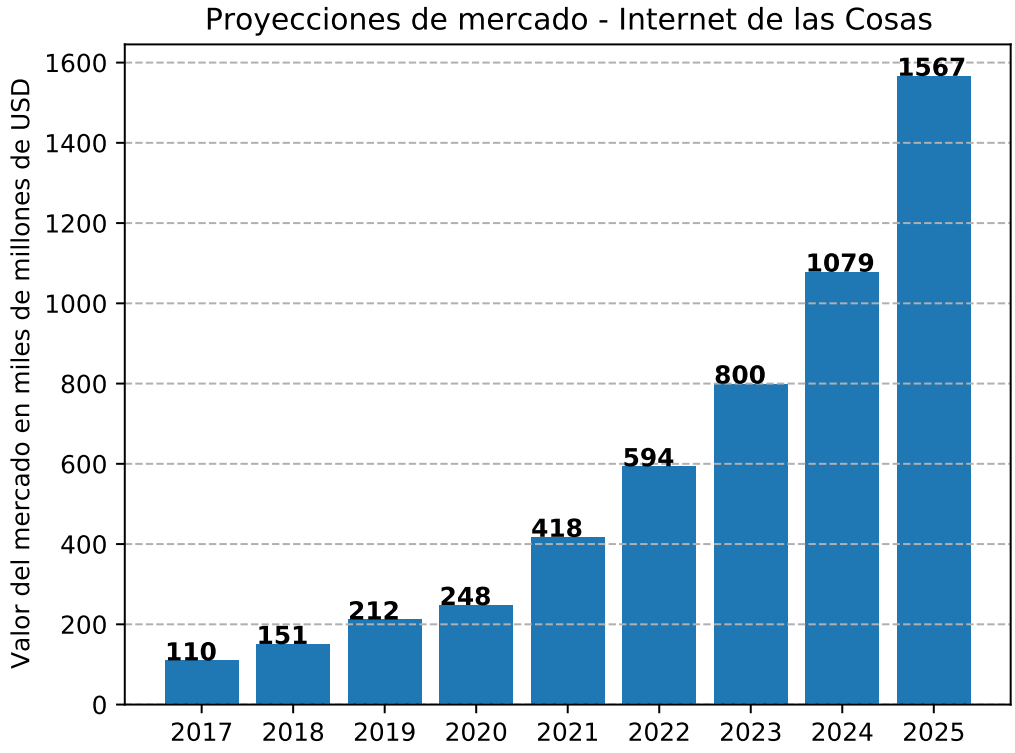

FIGURA 1.1: Proyecciones del valor de mercado a nivel mundial de la Internet de las Cosas.[1]

Por último, un punto que a menudo es pasado por alto al momento de hablar de la Internet de las Cosas, son todos los problemas asociados a la seguridad y la privacidad. Tener un mayor número de dispositivos conectados, muchos de los cuales recolectan información sumamente sensible, expande la superficie de ataque considerablemente, y deja tanto a empresas como a personas más vulnerables frente a posibles ciberdelincuentes. Existen incontables ejemplos de vulnerabilidades en dispositivos IoT que salieron a la luz, como una gigantesca *botnet* formada por dispositivos IoT [3], televisores inteligentes que espían conversaciones privadas [4], e incluso ataques dirigidos a dispositivos conectados en entornos industriales [5]. Estos hechos ocurren principalmente porque la seguridad y la privacidad a menudo son vistas como costos extra en los que no vale la pena incurrir. Incluso en ocasiones la propia empresa utiliza de manera poco ética esos

[1] Gráfico replicado en base al que se muestra en https://www.statista.com/statistics/976313/global-iot-market-size/.

agujeros de seguridad para obtener datos de los usuarios y sacar algún tipo de provecho económico a partir de ellos.

Las consecuencias de una falla de seguridad en un dispositivo IoT pueden ser muy graves, por lo que al momento de desarrollar una solución en el segmento del Internet de las Cosas, los aspectos de seguridad deben ser tenidos en consideración desde un principio, aún cuando ello implique desarrollar un producto más costoso o complejo.

1.2. Estado del arte

En la actualidad, existe una enorme cantidad de dispositivos conectados que pertenecen a muy diversos ámbitos, por lo que resultaría poco práctico brindar un panorama general del estado del arte en todos ellos. Por lo tanto, el foco estará en aquellos productos que ofrezcan soluciones orientadas a dispositivos que ya cuentan con cierto nivel de "inteligencia" (como maquinaria industrial/agrícola o artefactos del hogar), a los fines de dotarlos de conectividad y de todo un entorno que permita procesar y analizar los datos generados por ellos.

Por un lado, existen numerosas plataformas que solamente ofrecen la etapa de procesamiento, análisis y visualización de datos. Estas plataformas asumen que el cliente cuenta con el hardware necesario para enviarles la información relevante, y ellas se encargan de analizarla y presentarla de manera conveniente mediante tableros o *dashboards*, que le permiten al usuario ver el estado de los dispositivos y actuar sobre ellos. La mayoría de estas plataformas poseen características similares tales como administración de dispositivos, visualización de datos y soporte a múltiples protocolos de comunicación. Algunos ejemplos de esta clase de plataformas son ThingBoard [6], Thinger [7] y Ubidots [8], además de los servicios específicamente orientados a IoT de Amazon Web Services, de Google Cloud Platform y de Microsoft Azure.

Por otra parte, existen también plataformas de hardware que ofrecen soluciones genéricas de conectividad, junto con plataformas para analizar y visualizar los datos obtenidos, con características similares a las mencionadas anteriormente.

En este último segmento hay diferentes grados de especialización. Por ejemplo, existen empresas que ofrecen soluciones orientadas exclusivamente al ámbito industrial, como ThingWorx de la empresa PTC [9], mientras que otras brindan soluciones más genéricas con una plataforma de hardware y un entorno de análisis y visualización, como el caso de Particle (figura 1.2) [10].

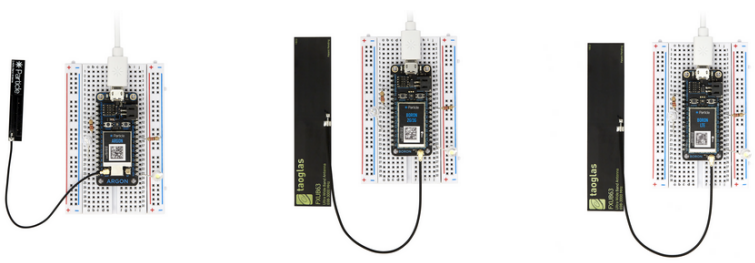

FIGURA 1.2: Plataformas de hardware ofrecidas por Particle.[2]

[2] Imagen extraída de https://store.particle.io/pages/prototyping-hardware.

1.3. Motivación

Un sector fundamental en el ámbito del Internet de las Cosas es el de la domótica, es decir el conjunto de tecnologías que permite el control y la automatización inteligente de una vivienda, a los fines de lograr una gestión eficiente y brindar mayor comodidad y seguridad a quienes la habitan [11].

A su vez, dentro de la domótica, un campo de especial interés es el de los electrodomésticos inteligentes (o *smart appliances*), tales como televisores, heladeras, hornos, lavarropas o microondas.

Una característica primordial de un electrodoméstico inteligente es su capacidad de estar conectado y ser manejado o accedido de forma remota, ya sea mediante una plataforma web, una aplicación en el celular, comandos de voz o cualquier otro medio. Esta capacidad brinda grandes ventajas con respecto a un artefacto convencional, ya que no solamente ofrece una mayor comodidad al momento de usarlo, sino que también permite programar acciones (como preparar un café a una determinada hora) y ahorrar energía al hacer más eficiente su uso.

Actualmente existen ya en el mercado numerosos ejemplos de electrodomésticos inteligentes que permiten llevar a cabo diferentes acciones:

- Encender el horno mediante un comando de voz, incluso con integración con los asistentes de voz más comunes.
- Consultar el contenido de la heladera mediante una aplicación en el celular.
- Recibir una notificación cuando el ciclo de lavado del lavarropas finaliza.
- Diagnosticar automáticamente la causa de una falla.

Gracias a las ventajas que ofrecen, la demanda de electrodomésticos inteligentes es cada vez más mayor, por lo que aquellas empresas o fabricantes que no se modernicen y comiencen a incorporar características *smart* a sus dispositivos, se encontrarán en clara desventaja al competir en el mercado con aquellas que sí lo hagan.

Como se puede apreciar en la sección 1.2, existen ya en el mercado numerosas soluciones orientadas a la Internet de las Cosas. Cada una de ellas ofrece diferentes características y cubre diferentes mercados. Dentro de las soluciones que ofrecen una plataforma tanto de hardware como de software, no existe un sector consolidado que esté orientado a dotar de conectividad a electrodomésticos en el hogar, y es allí donde surge la importancia del presente trabajo. Lo que se busca es cubrir justamente ese sector y ofrecer una solución personalizada a los fabricantes para que puedan lograr que los electrodomésticos que ya fabrican se conviertan en electrodomésticos inteligentes.

1.4. Objetivos y alcance

1.4.1. Objetivos generales

El objetivo general del presente trabajo es el diseño e implementación de un módulo capaz de dotar de conectividad WiFi y Bluetooth a un electrodoméstico convencional. Para ello, el módulo debe tener la capacidad de comunicarse con la placa del propio equipo y de recibir/enviar información a un servidor en la nube, a los fines de permitirle al usuario final un manejo remoto del aparato y conocer su estado, como así también enviar información de uso al fabricante.

El módulo no está destinado directamente al usuario final del electrodoméstico, sino a empresas o fabricantes que busquen incorporar características inteligentes a sus electrodomésticos.

A grandes rasgos, desde el punto de vista del usuario final, se debe permitir:

- Enviar por WiFi o Bluetooth un comando al electrodoméstico que dispare una acción en él, como iniciar la cocción en un horno o el lavado en un lavarropas.
- Conocer el estado del electrodoméstico mediante la recepción de información por WiFi o Bluetooth.

Por otra parte, desde el punto de vista del fabricante del electrodoméstico se debe permitir:

- Recibir y almacenar información acerca del estado de todos los dispositivos (si están o o no conectados, y en qué estado de ejecución se encuentran).
- Analizar y visualizar de manera conveniente la información de estado de los dispositivos a lo largo del tiempo.

1.4.2. Alcance

El presente trabajo incluye los siguientes aspectos:

1. Análisis, investigación y elección del hardware a utilizar en el módulo.
2. Implementación del firmware del sistema.
3. Comunicación con un microcontrolador que emule el comportamiento del electrodoméstico.
4. Desarrollo de una interfaz web mediante una plataforma ya existente, que permita enviarle comandos al electrodoméstico emulado.
5. Implementación de un entorno en la nube que permita analizar y visualizar los datos de los diferentes electrodomésticos conectados.

Es de especial importancia el punto 3 de la lista, en el que se define explícitamente que, a los fines de agilizar considerablemente el tiempo de desarrollo del trabajo, el prototipo no se utilizará en un electrodoméstico real. La interacción con él se emula mediante la comunicación con otro microcontrolador, que actúa como la placa de control del electrodoméstico: imita su comportamiento y sus respuestas ante diferentes estímulos.

En línea con el párrafo anterior, el presente trabajo no incluye lo siguiente:

1. Integración del prototipo a diferentes electrodomésticos/marcas con distintos tipos de comunicación serie y funcionalidades.
2. Desarrollo de una aplicación móvil desde la cual interactuar por Bluetooth con el módulo.
3. Diseño y fabricación de un circuito impreso. Para el trabajo se utiliza una placa de prototipo ya existente que incluye el hardware seleccionado.

Capítulo 2

Introducción Específica

En el presente capítulo se brinda una explicación del funcionamiento general del sistema implementado, y una introducción a diferentes tecnologías utilizadas en el trabajo. Se presentan además los requerimientos y la planificación del trabajo.

2.1. Funcionamiento general del sistema

Como se mencionó en el capítulo 1, el propósito del presente trabajo es el desarrollo de un módulo capaz de dotar de conectividad WiFi/Bluetooth a un electrodoméstico convencional. Para que ello sea posible, es necesario contar con diferentes módulos, tal como puede observarse en la figura 2.1, en la que se presenta un diagrama de bloques del sistema implementado.

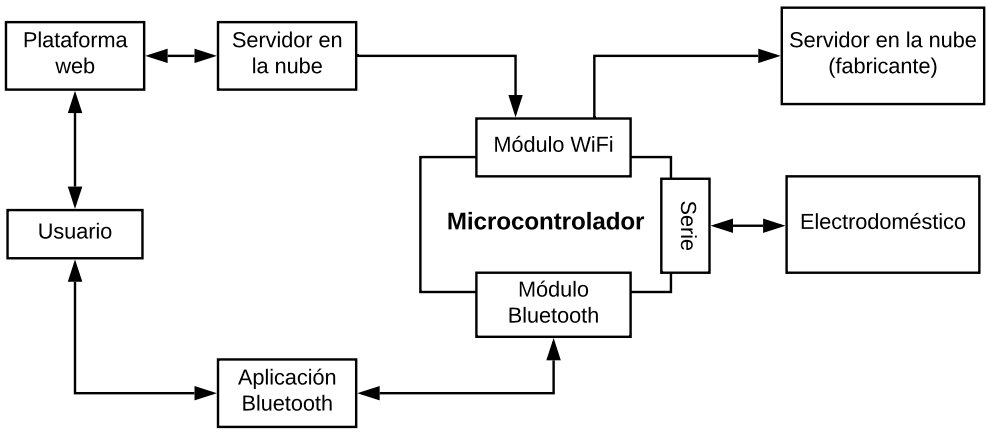

FIGURA 2.1: Diagrama general del sistema implementado.

El principal componente del sistema es el microcontrolador, que se encarga de procesar los comandos recibidos y de gestionar todas las comunicaciones, para lo que hace uso de sus interfaces de comunicación (módulos WiFi, Bluetooth y serie).

Con el fin de ilustrar el funcionamiento general del sistema, se presenta a continuación la serie de acciones que tiene lugar en un caso de uso típico del sistema, en el que el usuario desea que el electrodoméstico inicie una determinada operación (como por ejemplo, el ciclo de lavado en un lavarropas).

- El usuario final le envía un comando al electrodoméstico desde su computadora o celular.
  - En caso de que el comando sea enviado por WiFi, la comunicación pasa a través de un servidor en la nube que recibe el comando del usuario y luego lo envía al microcontrolador.
  - En caso de que el comando sea enviado por Bluetooth, la comunicación se realiza de manera directa con el microcontrolador.
- El módulo correspondiente (WiFi o Bluetooth, según sea el caso) recibe el comando y le informa al microcontrolador que hay un nuevo comando pendiente para procesar.
- El microcontrolador procesa el comando recibido y determina la acción a ejecutar.
- Si la acción a ejecutar lo requiere, el microcontrolador se comunica con la placa de control del electrodoméstico mediante una interfaz serie, a los fines de disparar en el mismo la operación deseada por el usuario.
- Dependiendo del tipo de acción, el electrodoméstico puede contestar con su información de estado. Esta es recibida por el microcontrolador y enviada de vuelta al usuario a través de la misma interfaz desde la cual recibió el comando originalmente. Es decir que si el usuario envió un comando por Bluetooth para consultar el estado del artefacto, el microcontrolador utiliza el módulo Bluetooth para devolverle la respuesta.

Cabe mencionar que debido a limitaciones de hardware, las interfaces de comunicación WiFi y Bluetooth no pueden ser utilizadas en simultáneo. Por lo tanto, si se envían comandos por WiFi, el Bluetooth debe estar apagado, y viceversa al enviar comandos por Bluetooth.

De manera independiente a las acciones disparadas a partir de un comando del usuario, el microcontrolador envía periódicamente al fabricante información acerca del estado del electrodoméstico. Este envío se lleva a cabo mediante la interfaz WiFi, que se comunica con un servidor en la nube solamente accesible por el fabricante, y que se encarga de almacenar la información para luego permitir su análisis y visualización. Gracias a esta información, el fabricante puede conocer y visualizar en todo momento el estado de sus electrodomésticos, ya sea de manera individual con todo el historial de uso de cada uno, o de forma general agrupando artefactos para obtener un panorama global del estado de sus dispositivos conectados.

Además, el módulo crea y mantiene activa en todo momento una red WiFi local de corto alcance en la que se encuentra corriendo un servidor web. El usuario puede conectarse a esta red y luego acceder desde cualquier navegador de Internet al servidor web, que permite configurar las credenciales de la red WiFi doméstica a la cual el módulo se conectará para recibir comandos y enviar información de uso al fabricante.

2.2. Tecnologías inalámbricas

En la actualidad, existen numerosas tecnologías inalámbricas que son aplicables a la Internet de las Cosas, muchas de las cuales surgieron debido a la relevancia que este concepto fue tomando, como es el caso de NB-IoT [12], LoRa [13] y SigFox [14]. Gran parte de estas tecnologías se caracterizan por permitir un bajo consumo de potencia y un largo alcance, a costa de una baja velocidad de transmisión. En muchas ocasiones esto es una relación de compromiso ideal para Internet de las Cosas.

A pesar del surgimiento de estas nuevas redes, las tecnologías tradicionales de conectividad inalámbrica, como el WiFi [15] y el Bluetooth [16], siguen siendo adecuadas para muchas aplicaciones, especialmente aquellas que requieren una interacción directa con el usuario final. Esto se debe a que son tecnologías ampliamente soportadas por la mayoría de los dispositivos que posee una persona, como computadoras y celulares.

Un electrodoméstico conectado interactúa de manera directa con la persona que lo utiliza en el hogar, por lo tanto el WiFi y el Bluetooth son tecnologías sumamente adecuadas para un módulo destinado a tal fin.

La tecnología WiFi permite transmitir a grandes velocidades, pero con un alcance bajo y un consumo de energía mayor. Esto último impide que el WiFi sea utilizado en, por ejemplo, sensores que funcionan a batería ubicados en lugares remotos, pero no supone un problema para un electrodoméstico que se encuentra en el hogar, con fácil acceso a una fuente de alimentación y cerca de un punto de acceso al cual conectarse.

Las especificaciones del WiFi definen una interfaz que se emplea para enviar y recibir señales entre un dispositivo inalámbrico (estación WiFi) y un punto de acceso. Si además se requiere tener acceso a Internet, es necesario conectarse también con un *router* y un módem, que a su vez debe estar conectado a un proveedor de servicios de Internet (ISP, por sus siglas en inglés correspondientes a *Internet Service Provider*). El módulo en el electrodoméstico actúa como una estación WiFi, es decir como un dispositivo que se conecta a la red doméstica de la casa y a través de ella obtiene una salida a Internet.

Por su parte, la tecnología Bluetooth pertenece a otro tipo de redes, denominadas Redes de Área Personal (PAN, por sus siglas en inglés). Una conexión Bluetooth permite la comunicación directa entre dos dispositivos cercanos y su uso está sumamente masificado, ya que es fácil y económico integrarlo en muchos aparatos. Estas son características que lo convierten en una tecnología ideal para utilizar en un electrodoméstico conectado.

Su utilización en el ámbito de la Internet de las Cosas cobró verdadera importancia gracias al surgimiento del Bluetooth Low Energy (BLE), que fue diseñado específicamente para proporcionar un bajo consumo de energía. Esto permitió incrementar aún más la popularidad de la tecnología, y extenderla hacia nuevos dispositivos como relojes o incluso zapatillas, lo cual fue acompañado con cada vez más modelos de computadores y celulares que también soporten la tecnología.

2.3. Protocolos HTTP/S y MQTT

Para que un dispositivo pueda conectarse a Internet, necesariamente debe recurrir al modelo TCP/IP [17], cuyas diferentes capas pueden observarse en la figura 2.2.

FIGURA 2.2: Capas del modelo TCP/IP.

El protocolo a nivel de capa de acceso a la red depende del hardware y del tipo de conectividad del dispositivo, y en el caso de este trabajo está constituido por la tecnología WiFi. Las capas de Internet y de transporte utilizan, en este trabajo, los protocolos que le dan el nombre al modelo, es decir IP (*Internet Protocol*) y TCP (*Transmission Control Protocol)*), respectivamente.

La capa de aplicación, ubicada en la parte superior del modelo, es la encargada de ofrecer a las aplicaciones de usuario la posibilidad de comunicarse con otros dispositivos a través de los servicios brindados por las demás capas.

El protocolo de aplicación más conocido es el Protocolo de Transferencia de Hipertexto (HTTP por sus siglas en inglés, correspondientes a *Hypertext Transfer Protocol*), que tiene una estructura cliente-servidor y permite realizar peticiones de datos y recursos [18]. Este protocolo es la base de cualquier intercambio de datos en la web, y por lo tanto es utilizado en aplicaciones de Internet de las Cosas cuando se desea que el dispositivo conectado acceda directamente a diferentes páginas web.

Existe una variante denominada Protocolo Seguro de Transferencia de Hipertexto (HTTPS por sus siglas en inglés, correspondientes a *Hypertext Transfer Protocol Secure*), que como su nombre lo indica, es una versión segura de HTTP en la que la transmisión está encriptada y el servidor, autenticado [19]. En toda aplicación, siempre se debe hacer lo posible para utilizar HTTPS y no simplemente HTTP, con el fin de garantizar la seguridad y la privacidad de los datos.

Además de los protocolos HTTP y HTTPS, existen otros para la capa de aplicación con características que los hacen ideales para la Internet de las Cosas, entre los que se destaca el protocolo MQTT (*Message Queue Telemetry Transport*) [20]. Este protocolo se basa en un modelo de publicaciones y subscripciones, en el que un cliente publica mensajes en un tema o *topic*, y todos aquellos nodos que se encuentran subscriptos a ese tema reciben el mensaje publicado. MQTT es ideal para aplicaciones de IoT, debido principalmente a que requiere un muy bajo ancho de banda, tiene un menor consumo de potencia que otras alternativas, y además es sencillo y ligero de implementar.

Por todo lo enunciado anteriormente es que se decide que el módulo desarrollado soporte los 3 protocolos: HTTP, HTTPS y MQTT.

2.4. Requerimientos

A continuación se presentan los requerimientos en base a los cuales se desarrolló el presente trabajo, agrupados en cuatro categorías.

1. Requerimientos generales del sistema.
   - *a*) El módulo debe ser capaz de llevar a cabo, mediante la recepción de comandos por WiFi o Bluetooth, las mismas funciones que a través de la interfaz física del electrodoméstico.
   - *b*) El módulo debe ser capaz de enviar comandos por WiFi o Bluetooth, para transmitir información de estado del electrodoméstico.
   - *c*) Las acciones a ejecutar de acuerdo al comando recibido dependen de cada aparato en particular, pero como mínimo se debe brindar la posibilidad de iniciar o detener la acción del electrodoméstico y consultar su estado.
   - *d*) El módulo debe enviar al fabricante información asociada al uso del electrodoméstico, e informar periódicamente el estado en el que se encuentra.
   - *e*) El módulo debe ser capaz de crear su propia red local WiFi a los fines de permitir configurar las credenciales de de la red WiFi a conectarse.
   - *f*) El módulo debe ser capaz de enviar y recibir comandos por WiFi utilizando los protocolos HTTP, HTTPS y MQTT.
2. Requerimientos de hardware.
   - *a*) El módulo debe poder comunicarse utilizando el Estándar IEEE 802.11 b/g/n (WiFi).
   - *b*) El módulo debe poder comunicarse utilizando Bluetooth Low Energy (BLE).
   - *c*) El módulo debe utilizar un único chip que integre el microprocesador y la conectividad WiFi/Bluetooth.
   - *d*) El módulo debe contar como mínimo con interfaces de comunicación serie SPI (*Serial Peripheral Interface*), I2C (*Inter-Integrated Circuit*) y UART (*Universal Asynchronous Receiver-Transmitter*), a los fines de poder adaptarse a los distintos tipos de electrodomésticos.
3. Requerimientos de firmware.
   - *a*) El firmware del módulo debe ser programado en lenguaje C.
   - *b*) Se deben realizar pruebas manuales para cada una de las funcionalidades del firmware del módulo.
4. Requerimientos de gestión de proyectos.

a) Se debe utilizar YouTrack [21] como herramienta de *issue tracking* y gestión de proyectos.

b) Se debe usar Git como sistema de control de versiones.

Cabe mencionar en este punto que originalmente se había planteado la integración del módulo a un electrodoméstico real, lo que implicaba también el diseño y fabricación de un PCB que se adaptase a él. Sin embargo, debido a las dificultades para acceder a un electrodoméstico sobre el cual probar el módulo, sumado a la necesidad de acelerar los tiempos de desarrollo, se decidió reemplazar dicho electrodoméstico por otro microcontrolador que emulase su comportamiento.

2.5. Planificación

Para mostrar la planificación del trabajo se recurre a un diagrama *Activity On Node* (figuras 2.3, 2.4 y 2.5). Allí cada caja o nodo representa una actividad, y las conexiones entre ellas representan una dependencia temporal en la que una debe terminarse antes que la siguiente. Además se muestra el tiempo en horas que demoraría cada una de las tareas. En la tabla 2.1 se explica el significado de cada color del diagrama.

TABLA 2.1: Código de colores del diagrama Activity On Node.

Color	Significado
Violeta	Planificación
Gris	Análisis e investigación
Rosa	Ingeniería de software
Naranja	Hardware
Rojo	Desarrollo de firmware
Verde claro	Pruebas
Celeste	Interfaz
Verde oscuro	Presentación

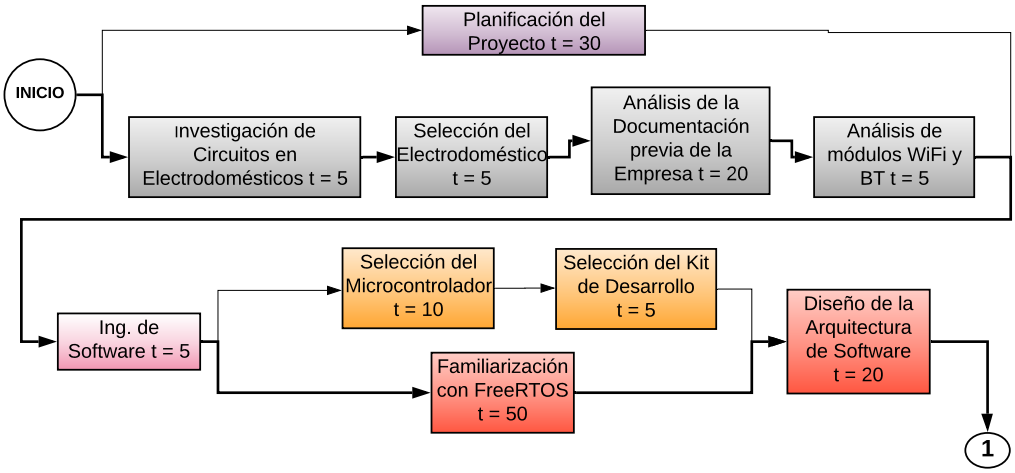

FIGURA 2.3: Diagrama Activity On Node parte 1.

Se puede observar que la primera etapa consiste en análisis e investigación, además de la planificación del trabajo. Luego sigue una etapa de diseño de firmware y familiarización con las herramientas a utilizar, sin empezar aún con la implementación.

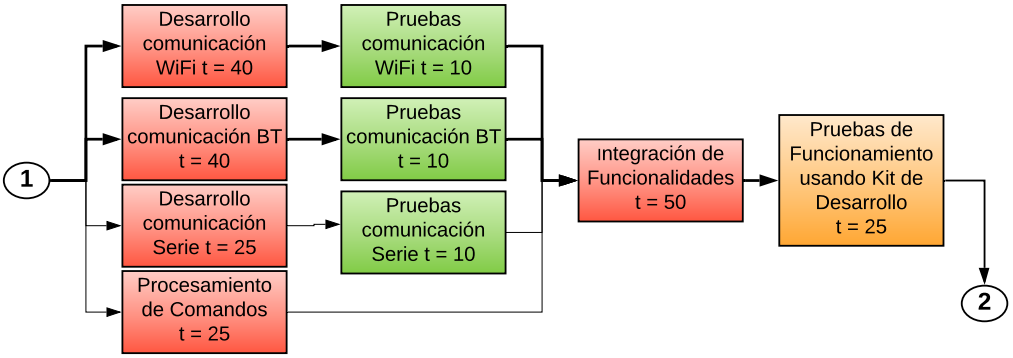

FIGURA 2.4: Diagrama Activity On Node parte 2.

Una vez definido el diseño, se procede con el desarrollo de las diferentes funcionalidades de firmware y sus respectivas pruebas. Si bien en el diagrama (figura 2.4) la realización de estas tareas se muestra en paralelo, ya que son relativamente independientes y por lo tanto podrían ejecutarse en simultáneo, al ser desarrollado por una única persona, las tareas se debieron desarrollar en serie.

Luego se integran todas las funcionalidades implementadas y se realizan pruebas generales del sistema, para posteriormente configurar las plataformas utilizadas para enviar y recibir información por WiFi y Bluetooth.

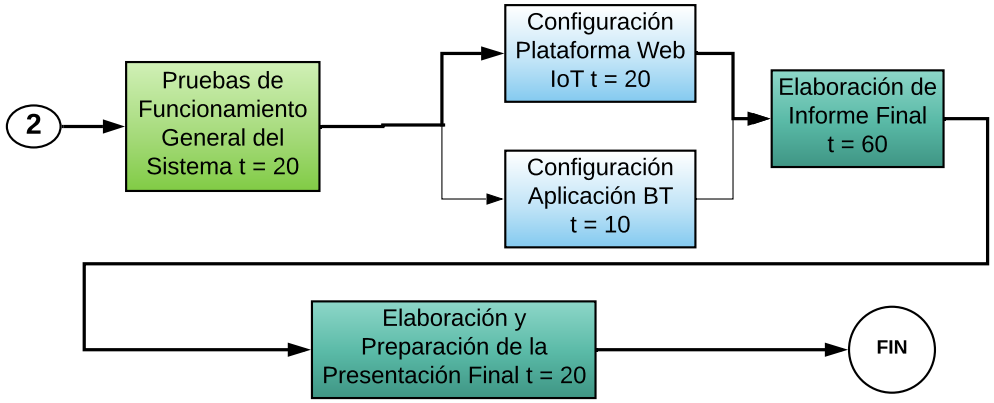

FIGURA 2.5: Diagrama Activity On Node parte 3.

Por último se contempla también el tiempo necesario para las actividades asociadas a la presentación del trabajo final.

Capítulo 3

Diseño e Implementación

En este capítulo se describe el diseño e implementación de los diferentes módulos de firmware del sistema, como así también las diferentes herramientas utilizadas. También se presenta la interfaz para el usuario final y cómo el módulo se integra con los servicios de plataformas en la nube.

3.1. Herramientas utilizadas

En esta sección se detallan las diferentes herramientas y tecnologías utilizadas, tanto a nivel de hardware como de software embebido, y se justifica su elección en cada caso.

3.1.1. Hardware

La principal elección a nivel de hardware radicó en el microcontrolador a utilizar en el módulo. Al momento de realizar la selección, se debieron tener en cuenta numerosos factores tales como capacidad de procesamiento, memoria RAM, memoria flash, cantidad de pines GPIO (*General Purpose Input Output*, es decir entradas y salidas de propósito general), interfaces de comunicación, consumo de energía, costos y disponibilidad, entre otros. Lógicamente, algunos factores son más importantes que otros dependiendo de la aplicación.

En el caso del módulo a desarrollar, los requerimientos planteados en la sección 2.4 impusieron restricciones considerables en cuanto a las interfaces de comunicación. Por un lado, debido a la comunicación con la placa de control del electrodoméstico, el microcontrolador debía contar al menos con interfaces de comunicación serie UART, I2C y SPI. Además, para simplificar el desarrollo y lograr un diseño más compacto, se exigió que el mismo chip cuente también con las interfaces para la comunicación WiFi y Bluetooth del módulo. Estos requerimientos en cuanto a interfaces redujeron considerablemente el abanico de posibilidades.

Por otra parte, al tratarse de un módulo que potencialmente tendría acceso a la misma fuente de alimentación que el electrodoméstico, el consumo de energía no era un factor importante a tener en cuenta al momento de la elección.

Debido a que el trabajo consistió en el desarrollo de un prototipo, era sumamente importante que el microcontrolador contase con un entorno de desarrollo robusto, tanto a nivel de placas o kits de desarrollo como a nivel de herramientas de

programación. Tener esto en cuenta al momento de la elección permitió un ahorro sustancial de posterior tiempo de desarrollo.

Por todo lo expuesto anteriormente, es que se decidió utilizar un ESP32 [22], el cual es un microcontrolador de 32 bits desarrollado por la compañía Espressif Systems y que cuenta con todas las interfaces de comunicación requeridas integradas en el mismo chip, tal como puede verse en la tabla 3.1.

TABLA 3.1: Interfaces de comunicación disponibles en un microcontrolador ESP32.

Interfaz de comunicación	Cantidad
WiFi	1
BLE	1
UART	3
SPI	4
I2C	2

Además, el microcontrolador ESP32 cuenta con un excelente entorno de programación, numerosas placas de desarrollo, documentación oficial de gran calidad y una comunidad muy activa.

En la figura 3.1 se puede observar la placa de desarrollo para ESP32 utilizada en el desarrollo de este trabajo, llamada NodeMCU [23]. Esta placa se eligió principalmente por su gran disponibilidad en el mercado local.

FIGURA 3.1: Placa de desarrollo para ESP32 NodeMCU.[1]

Cabe mencionar que la placa de desarrollo no utiliza directamente el microcontrolador ESP32, sino que usa el módulo ESP32-WROOM [24], el cual es un chip que posee en su interior un ESP32, pero contiene además una memoria flash de 4MB conectada a este y una antena de PCB integrada para la conectividad WiFi y Bluetooth, entre algunas otras funcionalidades.

[1]Imagen extraída de https://naylampmechatronics.com/espressif-esp/384-placa-de-desarrollo-para-esp32-nodemcu-32.html.

3.1.2. Software embebido

A nivel de software embebido, la principal decisión consistió en determinar si era necesario utilizar un sistema operativo de tiempo real (RTOS, por sus siglas en inglés correspondientes a *Real Time Operating System*).

Si bien el módulo no requiere cumplir con restricciones de tiempo real, un RTOS ofrece otras atractivas características que justifican su uso. Entre ellas, la de mayor utilidad es la capacidad de gestionar la ejecución de múltiples tareas. Además, ofrece otras herramientas que resultan de gran utilidad, como temporizadores o colas para comunicar tareas entre sí. También permite integrar con mayor facilidad librerías que implementen funcionalidades complejas como los protocolos TCP/IP.

Por ello es que, dada la complejidad del sistema a implementar, con múltiples tareas ejecutándose y comunicándose entre sí, se decidió utilizar un sistema operativo de tiempo real. La necesidad de recurrir a una implementación del *stack* TCP/IP contribuyó también a la decisión,

A su vez existen numerosas alternativas al momento de determinar qué RTOS utilizar. Sin embargo, el microcontrolador ESP32 ya cuenta con todo un entorno de desarrollo denominado ESP-IDF (*Espressif IoT Development Framework*), que es una versión de FreeRTOS [25] modificada para trabajar de manera óptima con este microcontrolador. Por ello se decidió utilizar este *framework* para el desarrollo del firmware y por lo tanto FreeRTOS como sistema operativo de tiempo real.

3.1.3. Plataforma en la nube

Para que el fabricante del electrodoméstico pueda almacenar, analizar y visualizar la información de sus diferentes electrodomésticos conectados, es sumamente conveniente recurrir a una plataforma de servicios en la nube. Estas plataformas se ocupan de gestionar toda la infraestructura necesaria asociada a la administración de dispositivos, autenticación, almacenamiento, análisis de datos y mucho más.

Existen diferentes alternativas de plataformas que se pueden usar para lograr este objetivo. Las más conocidas son las que ofrecen Google (Google Cloud Platform [26]), Amazon (Amazon Web Services [27]) y Microsoft (Microsoft Azure [28]). En los tres casos tienen servicios similares orientados específicamente a la Internet de las Cosas: Google Cloud IoT Core, AWS IoT y Azure IoT, respectivamente. Un gran atractivo de las tres, es que ofrecen un saldo inicial considerable que permite utilizar sus servicios sin necesidad de incurrir en gastos. Además, muchos servicios pueden ser utilizados sin costo indefinidamente, con ciertas restricciones en cuanto a recursos disponibles y con un consumo que debe mantenerse por debajo de ciertos valores. Esto permite probar las tres plataformas y atravesar toda la etapa de desarrollo y prototipado sin que ello represente un costo adicional.

Tras llevar a cabo un exhaustivo análisis para determinar cuál de las tres plataformas utilizar, y debido a que las tres plataformas cuentan con características muy similares, se decidió utilizar Google Cloud Platform principalmente por los siguientes motivos:

- La autenticación de los dispositivos es más sencilla y utiliza mecanismos estándar.
- La curva de aprendizaje inicial es menos pronunciada, lo que permite agilizar el desarrollo.
- Los dispositivos se pueden comunicar tanto por HTTP/S como por MQTT.

Al utilizar Google Cloud Platform, se hace también un uso exhaustivo de Google Cloud IoT Core, un servicio completamente administrado que permite conectar, administrar y transferir datos con rapidez y seguridad desde millones de dispositivos (o desde tan sólo unos pocos) en todo el mundo [29]. Además, Google Cloud IoT Core puede combinarse con otros servicios de la plataforma, para así tener a disposición una solución completa para recopilar, procesar, analizar y visualizar datos de IoT en tiempo real.

3.2. Firmware

Para implementar las funcionalidades explicadas en la sección 2.1, fue necesario desarrollar a nivel de firmware diferentes módulos que interactúan, como puede verse en el diagrama de la figura 3.2. En la arquitectura planteada, cada uno de los módulos del diagrama se corresponde con una tarea que se ejecuta en el sistema operativo de tiempo real.

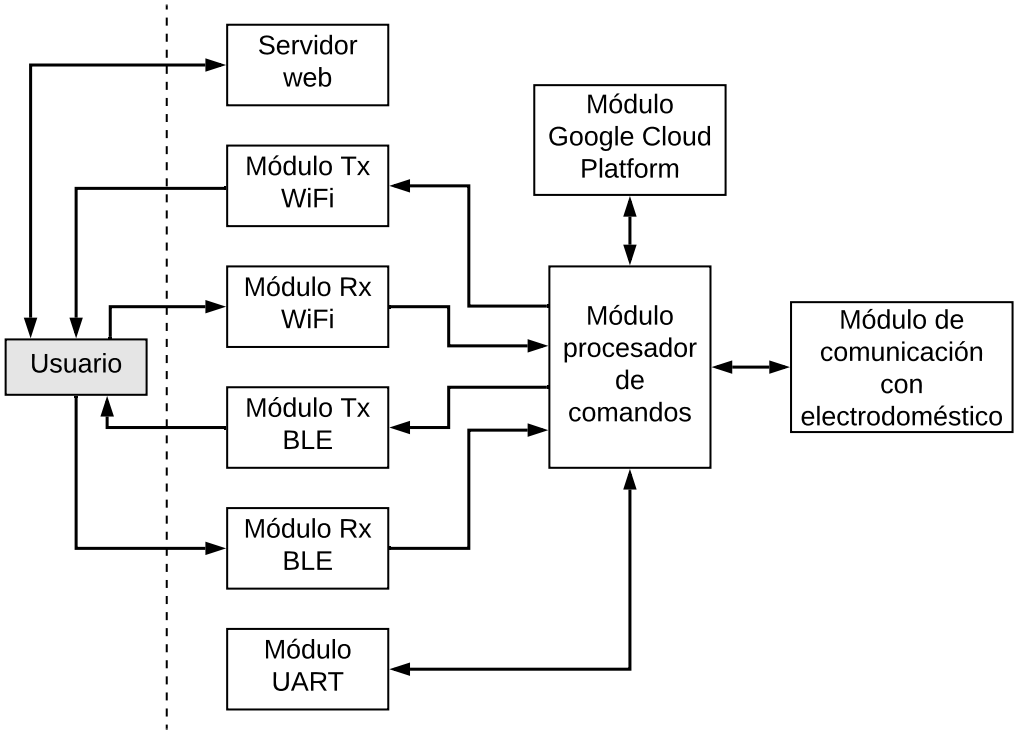

FIGURA 3.2: Diagrama de bloques de los módulos de firmware implementados.

De esta forma se tiene un módulo central (módulo procesador de comandos en el diagrama), que se encarga de recibir y procesar todos los comandos, y de generar

todas las respuestas necesarias, abstrayéndose del origen o destino de estos. Es decir que si el usuario del electrodoméstico manda un determinado comando por WiFi o Bluetooth, el módulo procesador de comandos es quien lo recibe en ambos casos, y luego se encarga de enviar la respuesta al módulo correspondiente (WiFi o Bluetooth según sea el caso). Además, este módulo es el único que interactúa con el módulo encargado de la comunicación con el electrodoméstico, por lo que toda comunicación con este debe pasar necesariamente a través del procesador de comandos. Esta centralización permite tener un gran control acerca de cómo ocurre la interacción de los módulos y las prioridades al momento de atender más de una consulta. La sincronización y el pasaje de datos entre las diferentes tareas de cada módulo y el procesador de comandos se llevan a cabo mediante colas o *queues* [30].

Por otra parte, se tienen los módulos encargados tanto de recibir los comandos enviados por el usuario y enviarlos al módulo procesador de comandos, como así también de recibir las respuestas para enviarlas nuevamente al usuario. Estos son los módulos Tx (transmisor, es decir, el que envía la respuesta al usuario) y Rx (receptor, es decir, el que recibe el comando del usuario), tanto WiFi como Bluetooth. En secciones posteriores se profundiza más sobre la implementación específica de estos módulos, pero a grandes rasgos se ocupan de manejar toda la interacción necesaria con el hardware y los *drivers* asociados a la comunicación WiFi/Bluetooth. También se encargan de detectar el comando/respuesta recibido y enviarlo a donde corresponda.

Cabe mencionar que en el caso de los módulos WiFi, se cuenta con tres implementaciones diferentes, una para cada uno de los protocolos de aplicación soportados: HTTP, HTTPS y MQTT. Si bien en los tres casos se reutiliza en su totalidad todo lo asociado a la interacción con el hardware, la forma de enviar y recibir información difiere considerablemente. Además, sólo uno de los tres protocolos se puede soportar en simultáneo, debido principalmente a limitaciones de memoria RAM, que impide asignar el *stack* suficiente para todas las tareas a la vez. Por ello es que mediante banderas del compilador (*compiler flags*) se especifica qué protocolo usará la imagen de firmware generada.

En la figura 3.2 también se observa un módulo UART, que a diferencia de los otros módulos de transmisión y recepción no interactúa con el usuario. Esto se debe a que es una tarea que se incluyó sólo con fines de *debug*, ya que permite enviar y recibir los mismos comandos que a través de las interfaces inalámbricas, pero de una forma mucho más simple utilizando una de las interfaces UART del microcontrolador.

Como se mencionó en la sección 3.1.3, para que el fabricante del electrodoméstico pueda analizar y visualizar la información de sus electrodomésticos conectados, se utiliza Google Cloud Platform. El módulo desarrollado envía periódicamente información acerca de su estado, para que luego esta sea almacenada, analizada y visualizada a lo largo del tiempo. De esto precisamente se encarga el módulo Google Cloud Platform. Cada vez que la tarea requiere obtener el estado del electrodoméstico, lo hace a través del procesador de comandos, enviando un comando de manera muy similar a como lo haría el usuario.

Otro de los módulos involucrados en la comunicación WiFi es el del servidor web. Esta tarea se encuentra corriendo permanentemente de fondo, y permite que el

usuario, conectándose a una red WiFi local generada por el propio microcontrolador (que actúa también como punto de acceso), ingrese a una página web para configurar las credenciales de la red WiFi del hogar a la cual el módulo se debe conectar.

Finalmente, se tiene el módulo de comunicación con el electrodoméstico. Esta tarea se encuentra bloqueada la mayor parte del tiempo, a la espera de que el procesador de comandos se comunique con ella, con el objetivo de disparar alguna acción en el electrodoméstico. Una vez que lo hace, la tarea se comunica con el artefacto mediante la interfaz serie correspondiente. Dado que el electrodoméstico se emula mediante otro microcontrolador que cuenta con una interfaz I2C, esta es la que se emplea para la comunicación serie.

3.2.1. Comunicación WiFi

Para lograr una comunicación con una página web externa en Internet desde el microcontrolador, es necesario contar con implementaciones de todas las capas del modelo TCP/IP. Para ello, se utilizaron en su mayor parte librerías ya existentes optimizadas para sistemas embebidos, tal como puede observarse en la tabla 3.2.

TABLA 3.2: Capas del modelo TCP/IP con sus correspondientes protocolos e implementaciones en el microcontrolador.

Capa del modelo TCP/IP	**Protocolo**	**Implementación**
Aplicación	HTTP/HTTPS/MQTT	ESP-IDF
Seguridad	TLS	mbed TLS
Transporte	TCP	lwIP
Internet	IP	lwIP
Acceso a la red	WiFi	ESP-IDF

Para manejar la comunicación a más bajo nivel mediante WiFi, se recurrió al *driver* que forma parte del *framework* ESP-IDF desarrollado por el propio fabricante del microcontrolador. Este *driver* expone una API con ciertas funciones que el programador puede llamar para interactuar con la interfaz WiFi del chip. También fue necesario definir un *event handler* (manejador de eventos), es decir, una función que el *driver* WiFi llama cada vez que debe notificar un nuevo evento que haya ocurrido. En la figura 3.3 se puede apreciar esta interacción entre el *driver* y el programa principal.

El *driver* WiFi permite utilizar la interfaz en tres modos diferentes:

- Modo estación (*station mode*) o modo cliente WiFi, en el que el microcontrolador se conecta a un punto de acceso.
- Modo punto de acceso (*AP mode*), en el que el microcontrolador actúa como un punto de acceso al que otros clientes se conectan.
- Modo combinado (*combined AP-STA mode*), en el que el microcontrolador se comporta como punto de acceso y a la vez como una estación que se conecta a otro punto de acceso.

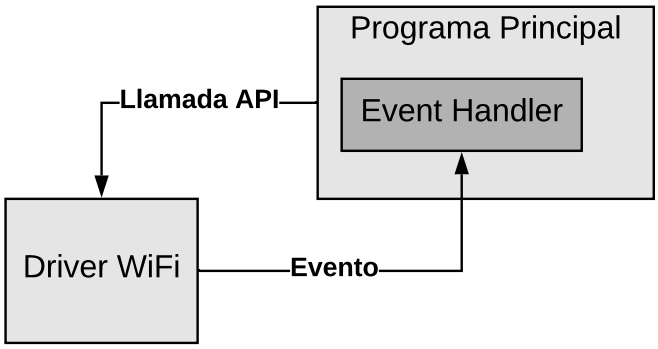

FIGURA 3.3: Interacción entre el driver WiFi y el programa principal.

En este trabajo se utilizó el modo combinado, ya que se requiere que el chip funcione como estación (para conectarse a la red WiFi del hogar que le da salida a Internet) y como punto de acceso (para que el usuario pueda acceder al servidor web que corre en el microcontrolador).

Con respecto al *event handler*, los eventos de interés que gestiona son los siguientes:

- STA_START: se dispara luego de que el driver se ha inicializado como estación. En este punto es posible conectarse a una red en particular.
- STA_GOT_IP: se dispara cuando se logra una conexión exitosa a una red y se obtiene una dirección IP.
- STA_DISCONNECTED: se genera cuando se produce una desconexión de la red.
- AP_START: se dispara luego de que el driver se ha inicializado como punto de acceso. En este punto se puede comenzar a ejecutar el servidor web.
- AP_STACONNECTED: se produce cuando una estación se conecta al punto de acceso.
- AP_STADISCONNECTED: se genera cuando una estación se desconecta del punto de acceso.

Muchas tareas necesitan tener conocimiento de los eventos que gestiona el *event handler*, principalmente requieren conocer cuándo el módulo ha logrado conectarse a la red deseada. Para lograr esta sincronización se utilizaron *event groups* [31], cuyos bits se programan en el *event handler* del *driver* WiFi. Por ejemplo, cuando se establece una conexión exitosa a una red, el bit 0 del *event group* se pone en 1 y así se disparan de manera sincronizada aquellas tareas que estaban esperando que se establezca dicha conexión.

Continuando con las próximas capas del modelo, para las de Transporte/Internet, se hizo uso de los protocolos TCP/IP. Para ello se utilizó la librería lwIP [32], que es una implementación de código abierto de los protocolos TCP/IP diseñada para sistemas embebidos, ya que busca minimizar al máximo la utilización de recursos.

El protocolo TCP/IP no proporciona ningún tipo de seguridad para la red en la que se los utiliza. Sin embargo, es posible agregar una capa adicional por encima de la de transporte, que permita establecer una comunicación segura, encriptada y autenticada a lo largo de una red no segura. Esta seguridad adicional para la capa de transporte se denomina *Transport Layer Security* (TLS) [33] y para su implementación se recurrió a la librería mbed TLS [34].

Finalmente, para la capa de aplicación se utilizaron tres protocolos diferentes: HTTP, HTTPS y MQTT. En el caso de HTTPS y MQTT la comunicación es segura ya que se utiliza también TLS, mientras que para el protocolo HTTP la información no se transmite de manera segura y podría ser interceptada en el camino.

Como ya se mencionó anteriormente, los tres protocolos no son soportados en forma simultánea, sino que en tiempo de compilación se decide cuál de ellos se va a usar en la imagen generada.

En el caso de los protocolos HTTP y HTTPS, si bien el *framework* ESP-IDF ya cuenta con implementaciones de clientes, se decidió llevar a cabo una implementación propia a los fines de profundizar los conocimientos en la temática. Para ello, en el cliente HTTP se utilizó la librería lwIP con el fin de crear los *sockets* necesarios y realizar las escrituras y lecturas necesarias, creando métodos para enviar una HTTP *request* y recibir una HTTP *response*. El caso del cliente HTTPS es más complejo, y se utilizó la librería mbed TLS para crear todas las estructuras necesarias para la transacción segura, y luego también se definieron métodos para enviar *requests* y recibir *responses*, incluyendo el proceso del *handshake* TLS [35].

Una vez implementados los clientes HTTP y HTTPS, se los utilizó en los respectivos módulos de transmisión y recepción de comandos por WiFi. Las tareas implementadas son relativamente sencillas y muy similares para ambos protocolos, y se diferencian principalmente en las funciones que llaman para enviar y recibir información. En la figura 3.4 se puede apreciar un diagrama de flujo de la tarea de transmisión por WiFi usando HTTP/HTTPS. Cabe mencionar que el que envía datos nuevos a la cola de la tarea es el procesador de comandos, cuando desea devolver una respuesta al usuario.

Por otra parte, en la figura 3.5 se observa un diagrama de flujo de las tareas de recepción de comandos empleando HTTP/HTTPS. El funcionamiento básico de esta tarea consiste en consultar periódicamente si el usuario ha enviado un nuevo comando, y en caso afirmativo reenviarlo al procesador de comandos.

Finalmente, para el caso de los módulos WiFi con protocolo MQTT, se decidió utilizar directamente las implementaciones del protocolo que ya forman parte del *framework* ESP-IDF, debido a que una implementación propia resultaría demasiado compleja y escapa al alcance de este trabajo. El funcionamiento general de las tareas es análogo al de las versiones con HTTP/HTTPS, con la diferencia de que ahora se utiliza MQTT para publicar las respuestas hacia el usuario en un *topic* determinado, y recibir comandos desde otro *topic* al cual el microcontrolador se encuentra subscripto. El protocolo MQTT se utilizó sobre TLS, por lo tanto todos los intercambios de información ocurren de manera segura.

De forma similar al *driver* WiFi, se implementó un *event handler* que gestiona los siguientes eventos asociados al protocolo MQTT:

- MQTT_EVENT_CONNECTED: se dispara cuando se logra una conexión exitosa con el *broker* MQTT.

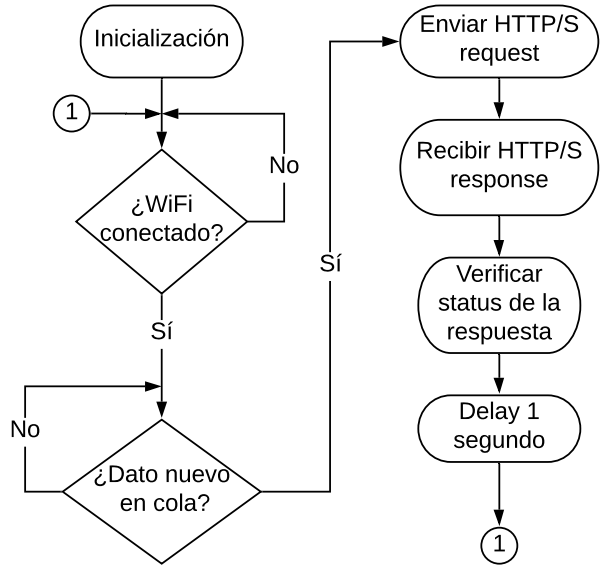

FIGURA 3.4: Diagrama de flujo para las tareas de transmisión por HTTP/HTTPS.

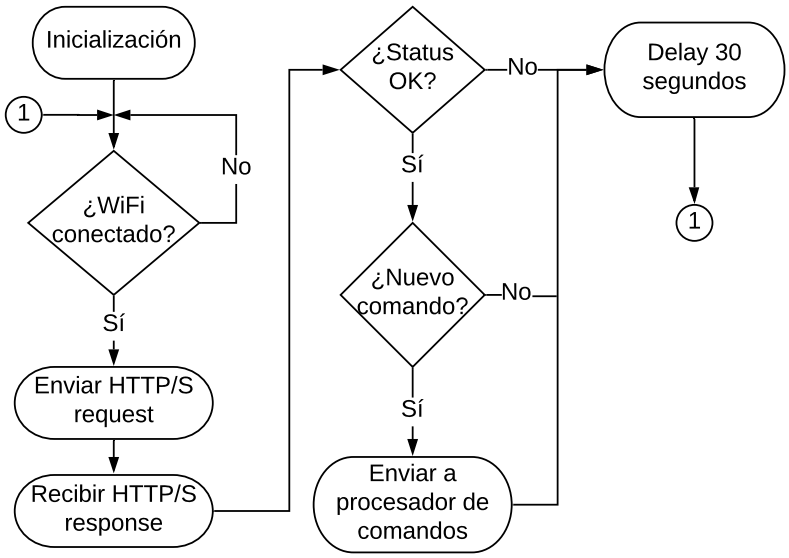

FIGURA 3.5: Diagrama de flujo para las tareas de recepción por HTTP/HTTPS.

- MQTT_EVENT_DISCONNECTED: se genera ante una desconexión del *broker*.
- MQTT_EVENT_SUBSCRIBED/UNSUBSCRIBED/PUBLISHED: se produce cada vez que un cliente MQTT se suscribe, elimina la suscripción o publica datos en un *topic* determinado.
- MQTT_EVENT_DATA: se genera cada vez que un *topic*, al que algún cliente se encuentra suscripto, recibe información nueva y la envía al cliente MQTT.

3.2.2. Comunicación BLE

Un dispositivo BLE puede actuar como *peripheral* (periférico o servidor) o como *central* (principal o cliente), y en el caso de este trabajo el microcontrolador actúa como un *peripheral*. El servidor anuncia continuamente su presencia, enviando paquetes para que otros dispositivos detecten su presencia. Por su parte, el dispositivo *central*, que en este caso sería la aplicación en el celular del usuario del electrodoméstico, realiza un escaneo en busca de dispositivos cercanos. Una vez que encuentra el *peripheral* deseado, establece una conexión y le realiza diferentes peticiones.

De manera similar al caso del WiFi, se cuenta con un *driver* que gestiona la interfaz Bluetooth y se comunica con el programa principal mediante eventos. En este caso se deben definir varios *event handlers* para gestionar diferentes facetas de la comunicación. Para entender el funcionamiento de la comunicación BLE implementada, es necesario comprender también algunos conceptos asociados al propio protocolo.

Por un lado se tiene el concepto de GAP (*Generic Access Profile*), que controla las conexiones y el *advertising*, es decir, cómo el servidor anuncia su presencia transmitiendo paquetes periódicamente. Mediante las estructuras de datos y funciones que el *framework* expone, se especifican diferentes parámetros como el *advertising interval* (cada cuánto el dispositivo anuncia su presencia) o el nombre del dispositivo, entre muchos otros. Además se tiene un *event handler* para los eventos asociados al GAP, entre los que se destacan dos en particular:

- GAP_BLE_ADV_DATA_SET_COMPLETE_EVT: se dispara cuando los parámetros de *advertising* se encuentran configurados, y por lo tanto se lo utiliza para iniciar el proceso de *advertising*. Es decir que cuando este evento se detecta, el dispositivo comienza a anunciar su presencia.
- GAP_BLE_ADV_START_COMPLETE_EVT: este evento indica que el proceso de *advertising* terminó su inicialización, y se lo puede utilizar para determinar si dicha inicialización tuvo éxito u ocurrió alguna falla.

En cuanto a la forma de transmitir datos, el intercambio de información entre dos dispositivos BLE se realiza mediante los denominados atributos genéricos o GATT (*Generic Attributes*), estructuras de datos jerárquicas que constituyen la base del protocolo. Esta jerarquía puede observarse en la figura 3.6 y está constituida por varias partes:

- Perfil (*profile*): es la jerarquía de mayor nivel y está formada por uno o más servicios.
- Servicio (*service*): es un conjunto de diferentes informaciones (como lecturas de un sensor), y contiene por lo menos una característica. Existen numerosos servicios predefinidos por el *Bluetooth Special Interest Group* (SIG), como presión sanguínea o ritmo cardíaco, y además se pueden crear servicios personalizados si el uso no está contemplado en esa lista predefinida, como es el caso de este trabajo.
- Característica (*characteristic*): es donde se encuentran los datos propiamente dichos. Posee varios campos, entre los que se destacan el de *value*, con el valor en sí, y el de *properties*, que describe cómo se puede interactuar con el valor.

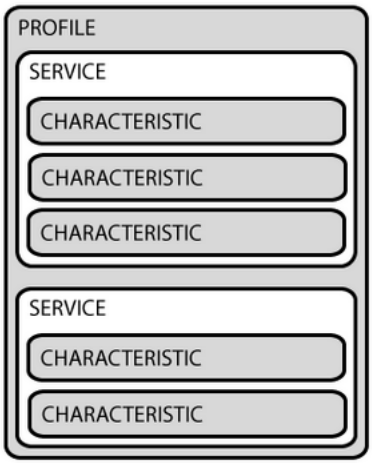

FIGURA 3.6: Jerarquía de la estructura GATT.[2]

Un cliente puede llevar a cabo diferentes operaciones sobre una característica de un *peripheral*, que incluyen operaciones de lectura y escritura. Esto justamente es lo que se utilizó para enviar comandos por BLE (se escribe una característica en particular) y para recibir respuestas (se lee una característica).

Se implementó también un *event handler* para los eventos asociados a GATT, entre los que se destacan los siguientes:

- GATTS_REG_EVT: es el primer evento que se dispara al iniciar el servidor BLE. Se lo utiliza para configurar los parámetros de *advertising* (lo que dispara luego el correspondiente evento en el *event handler* de GAP) y para crear el servicio que usa la aplicación.
- GATTS_CREATE_EVT: es el próximo evento en dispararse luego de que el servicio se crea exitosamente. Se lo usa para iniciar el servicio y para agregarle las características que luego se emplean para enviar comandos y recibir respuestas.
- GATTS_CONNECT_EVT/GATTS_DISCONNECT_EVT: se dispara cada vez que un cliente se conecta/desconecta del *peripheral*.
- GATTS_WRITE_EVT: se produce cuando el dispositivo cliente desea realizar una operación de escritura sobre una característica, es decir, cuando el usuario desea mandar un comando al electrodoméstico. En este punto se lee lo que el usuario escribió en la característica y se lo envía al procesador de comandos.
- GATTS_READ_EVT: se produce cuando el cliente desea realizar una operación de lectura sobre una característica, es decir, cuando el usuario desea recibir una respuesta del electrodoméstico. Para ello se escribe en la característica correspondiente el valor que se recibe desde el procesador de comandos.

[2] Imagen extraída de https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

3.2.3. Procesamiento de comandos

La tarea que se encarga del procesamiento de comandos se encuentra bloqueada hasta que algún módulo escriba en la cola correspondiente. Se utiliza una única cola a la que todos los módulos envían datos, a través de una estructura que incluye no solamente el comando propiamente dicho, sino también un identificador del módulo que escribió en la cola, tal como puede verse en el algoritmo 3.1.

```
/*-------------------------------------------------------------------
|  Struct: rx_command_t
| -------------------------------------------------------------------
|  Description: represents a received command.
|
|  Members:
|       rx_id   - module that sent the command
|       command - command received
*-------------------------------------------------------------------*/
typedef struct {
    rx_module_t     rx_id;
    command_type_t  command;
}   rx_command_t;
```

ALGORITMO 3.1: Pseudocódigo del lazo principal de control.

Cuando se necesita enviar una respuesta, se utiliza ese *rx_id* para determinar el módulo adecuado y, por lo tanto, en qué cola se debe escribir la respuesta.

El tipo de dato *command_type_t* define los diferentes comandos disponibles, y es fácilmente extensible con nuevos valores. Existen dos grupos de comandos:

- Comandos dirigidos al módulo: no se comunican con el electrodoméstico, sino que actúan solamente sobre el módulo de conectividad.
  - CMD_WIFI: se utiliza para cambiar el estado de la conexión WiFi (si está apagada la enciende, si está encendida la apaga). Debido a que el WiFi y el BLE no pueden estar funcionando en simultáneo, en caso de que este último se encuentre encendido, se lo desactiva antes de iniciar la conexión WiFi.
  - CMD_BLE: análogo a CMD_WIFI, pero aplicado a Bluetooth Low Energy.
  - CMD_ECHO: comando utilizado para *debug*, simplemente le devuelve a la misma interfaz los mismos datos que envió.
- Comandos dirigidos al electrodoméstico: son una serie de comandos genéricos a modo de ejemplo, los cuales se envían por I2C al microcontrolador que emula al electrodoméstico.
  - CMD_SLAVE_START_A/CMD_SLAVE_START_B: inicia en el electrodoméstico un determinado proceso.
  - CMD_SLAVE_PAUSE: pausa el proceso actual.
  - CMD_SLAVE_CONTINUE: reanuda un proceso pausado.
  - CMD_SLAVE_RESET: reinicia el electrodoméstico.
  - CMD_SLAVE_STATUS: se lo emplea para preguntar por el estado actual del electrodoméstico.

Para el microcontrolador que emula el comportamiento del electrodoméstico, que también es un ESP32, se desarrolló un firmware simplificado con una tarea principal que consiste en una máquina de estados, la cual representa los diferentes estados del electrodoméstico. Los comandos que recibe por su interfaz I2C pueden modificar esta máquina de estados, lo que refleja que un determinado proceso inicia o termina. Además, cada vez que recibe un comando desde el módulo principal, devuelve una señal de *acknowledge*, la cual es verificada por la tarea de procesamiento de comandos cada vez que se produce una comunicación con el electrodoméstico.

3.3. Integración con Google Cloud Platform

Como se mencionó en la sección 3.1.3, para proporcionarle información al fabricante acerca del estado del electrodoméstico se utiliza Google Cloud Platform. Esta implementación tiene dos facetas: por un lado, la tarea a nivel de firmware, que permite que el microcontrolador se comunique con Google Cloud, y por el otro, toda la infraestructura y los servicios utilizados en Google Cloud.

Se utilizaron diferentes servicios de Google Cloud Platform, y el principal es Cloud IoT Core. Este servicio cuenta con dos componentes centrales:

- *Device manager* (administrador de dispositivos): permite registrar y administrar dispositivos, y ofrece también un mecanismo para autenticarlos cuando se conectan. El módulo desarrollado se registra aquí como dispositivo.
- *Protocol bridge* (puente de protocolos): permite acceder a Google Cloud de manera segura mediante protocolos MQTT y HTTP.

El servicio de Cloud IoT Core interactúa con varios otros servicios de Google Cloud para armar un flujo de análisis completo, como puede observarse en la figura 3.7.

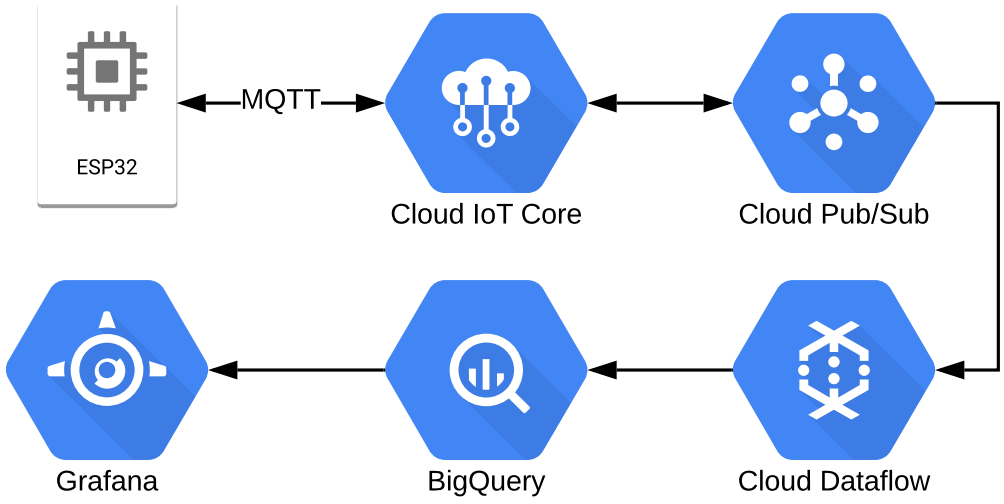

FIGURA 3.7: Arquitectura utilizada en Google Cloud Platform.

El servicio más importante con el que interactúa es Cloud Pub/Sub, ya que es donde se publican todos los datos recibidos por el *protocol bridge*. De hecho, cada

dispositivo en Cloud IoT Core tiene asociado al menos un *topic* en Cloud Pub/-Sub, y es en ese *topic* donde el microcontrolador realiza la publicación de los datos que desee.

En este punto ya se tiene almacenada toda la información cruda enviada por el electrodoméstico, y los bloques restantes se encargan de mostrar gráficamente esa información utilizando Grafana, una herramienta que permite la visualización de datos provenientes de distintas fuentes. En este caso se utiliza como fuente Big Query, una base de datos altamente eficiente y escalable ofrecida por Google Cloud. Para que los datos de los *topics* se transfieran y almacenen automáticamente a la base de datos, se utiliza Cloud Dataflow. Finalmente se emplea la propia infraestructura de Google Cloud para levantar la aplicación de Grafana, y así pueden visualizarse los datos enviados por el dispositivo.

Por otro lado, a nivel de firmware se cuenta con una tarea dedicada exclusivamente a enviar de manera periódica a Google Cloud información acerca del estado actual del electrodoméstico. Esta transmisión se lleva a cabo solamente utilizando el protocolo MQTT, es decir que no hay soporte para HTTP/HTTPS. ,

El principal desafío para enviar la información está en las autenticaciones necesarias para que esto sea posible.

Debido a que la transmisión es segura (utiliza TLS por debajo) es necesario utilizar los certificados de Google Cloud Platform al momento de configurar el cliente MQTT. Esto no difiere del procedimiento realizado para las tareas de MQTT que interactúan con el usuario.

También se debe autenticar el dispositivo en particular, es decir que el microcontrolador debe demostrar que, cuando se comunica con Google Cloud, es el dispositivo que dice ser y no otro haciéndose pasar por él [36]. Para ello es necesario generar claves, asociarlas a cada dispositivo en Cloud IoT Core y luego almacenarla también en el firmware. La figura 3.8 ilustra este proceso.

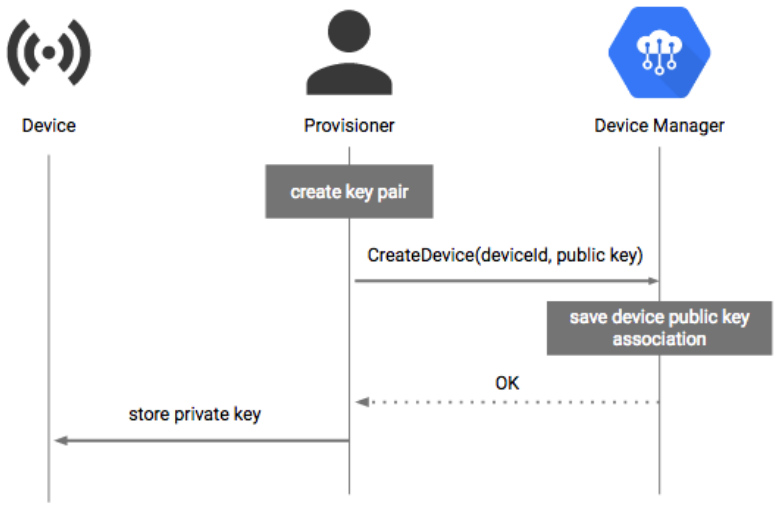

FIGURA 3.8: Proceso de generación de claves para el servicio de Cloud IoT Core.[3]

[3] Imagen extraída de https://cloud.google.com/iot/docs/concepts/device-security#provisioning_credentials.

Luego el firmware debe utilizar la clave que tiene almacenada para generar un JSON Web Token (JWT) [37] y enviarlo al momento de conectarse por MQTT, como se observa en la figura 3.8.

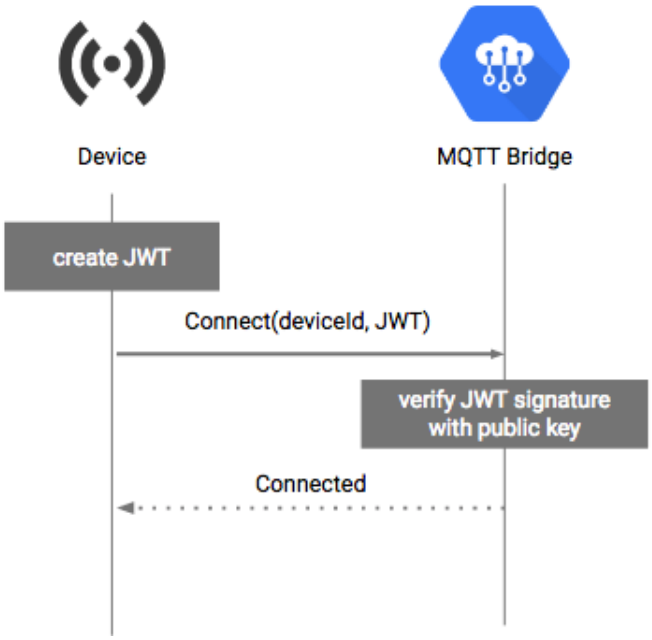

FIGURA 3.9: Autenticación del dispositivo utilizando un JWT.[4]

La generación del JWT tiene la complejidad adicional de que es necesario conocer la fecha y hora actual, ya que son parte de los campos que se deben enviar para autenticarse y para configurar una fecha de expiración para el *token*. Debido a que el dispositivo ya cuenta con conexión a Internet, para determinar la hora actual se recurrió al Protocolo Simple de Hora de Red (SNTP, por sus siglas en inglés correspondientes a *Simple Network Time Protocol*), el cual permite que un cliente obtenga información horaria desde un servidor de tiempo concreto. En este caso se utilizó pool.ntp.org como servidor horario, debido a que es un proyecto gratuito muy utilizado a nivel mundial y con servidores distribuidos globalmente.

[4] Imagen extraída de https://cloud.google.com/iot/docs/concepts/device-security#provisioning_credentials.

Capítulo 4

Ensayos y Resultados

En este capítulo se presentan las pruebas realizadas para validar el correcto funcionamiento del sistema implementado. Se incluyen pruebas funcionales de cada bloque en particular y de todo el sistema integrado.

4.1. Pruebas funcionales

A medida que se desarrollaron los diferentes módulos del firmware del sistema, se llevaron a cabo las correspondientes pruebas en cada uno de ellos, para validar que funcionaban de la manera esperada antes de ser integrados con otras partes del sistema. Todas las pruebas fueron realizadas manualmente.

Los bloques a probar se pueden dividir en tres grupos: la comunicación por WiFi, la comunicación utilizando BLE y la comunicación con el electrodoméstico.

4.1.1. Comunicación WiFi

La comunicación por WiFi es la parte del sistema más compleja y la que por lo tanto requiere pruebas más exhaustivas de sus diferentes funcionalidades.

Independientemente de la aplicación que se le vaya a dar, el primer paso es lograr conectarse exitosamente a una red WiFi. Esta fue la primera prueba realizada, configurando a mano las credenciales de la red inalámbrica y verificando por consola que se conecte correctamente, como se puede ver en la figura 4.1. En ella, se ve que se inició la conexión y luego se registró el evento en el que el microcontrolador obtuvo una dirección IP y por lo tanto se conectó con éxito.

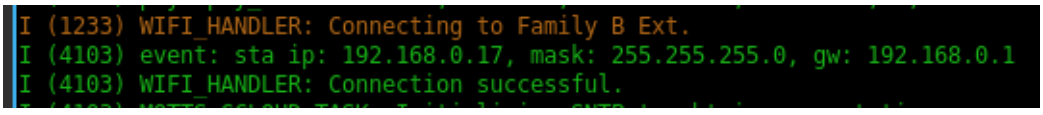

FIGURA 4.1: Salida por consola del microcontrolador al conectarse a una red WiFi.

Una vez llegado a este punto, es posible continuar con las pruebas asociadas a la comunicación con el usuario del electrodoméstico, mediante una interfaz de usuario, y con el fabricante, enviando información hacia Google Cloud.

Comunicación con el usuario

A los fines de contar con algún mecanismo que permita enviar comandos desde Internet de parte del usuario del electrodoméstico, se recurrió a diferentes plataformas web IoT configurables ya existentes.

Para las pruebas de envío y recepción mediante los protocolos HTTP y HTTPS, se utilizó ThingSpeak [38]. Esta plataforma permite recibir información desde el microcontrolador, almacenarla y graficarla. Además, permite enviar comandos desde la página hacia el microcontrolador. Si bien ThingSpeak ofrece una librería para interactuar con la plataforma, en este caso se utilizaron directamente las direcciones indicadas en su documentación para enviar HTTP *requests* desde las tareas de transmisión y recepción WiFi que se ejecutan en el microcontrolador. De esta forma, por un lado se enviaron comandos desde ThingSpeak y se observó por consola que el microcontrolador los recibía correctamente. Por otro lado se enviaron datos periódicamente hacia ThingSpeak y se los graficó, verificando que tenían los valores esperados.

Para demostrar la versatilidad del módulo desarrollado, se decidió utilizar una plataforma IoT diferente para probar la comunicación utilizando MQTT. Por eso se utilizó Adafruit IO [39], plataforma que permite crear tableros o *dashboards* personalizados para comunicarse con sistemas embebidos. La figura 4.2 muestra la interfaz desarrollada, en la que se puede ver que se tienen diferentes botones para mandar comandos determinados, un historial de comandos enviados, un cuadro mostrando el estado del electrodoméstico (*slave*) y un cuadro con el último comando enviado.

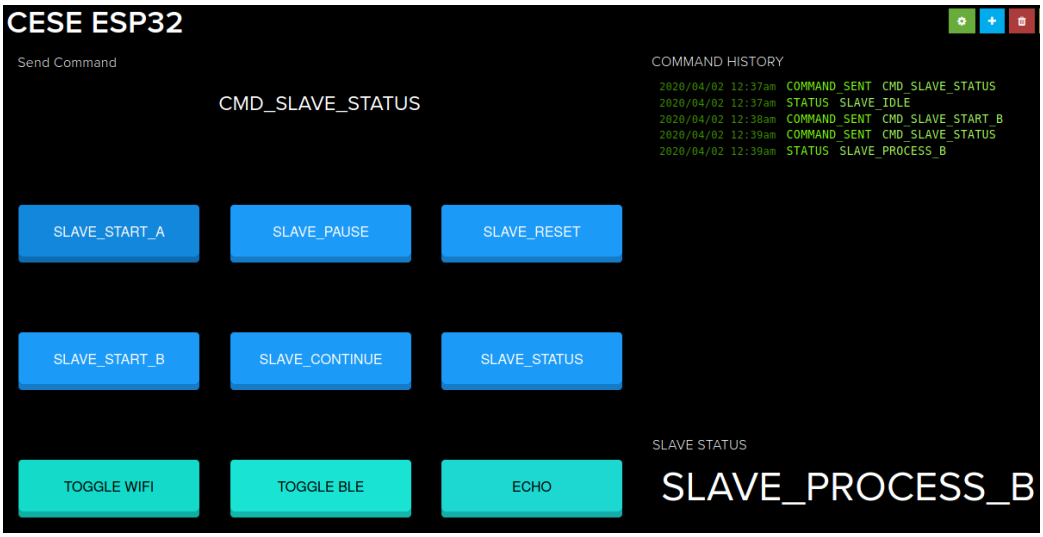

FIGURA 4.2: Interfaz de usuario creada en Adafruit IO.

En el historial de comandos se puede observar que se envió un comando para conocer el estado del electrodoméstico (CMD_SLAVE_STATUS) y se recibió de vuelta el estado actual (SLAVE_IDLE, es decir inactivo sin ejecutar ninguna acción). Luego se envió un comando para iniciar un proceso en el electrodoméstico (CMD_SLAVE_PROCESS_B), y al preguntar nuevamente por el estado, se obtuvo que se encontraba ejecutando la acción iniciada con el comando anterior.

Desde el punto de vista del firmware, en la figura 4.3 se observa cómo van ocurriendo los diferentes eventos asociados a MQTT (descriptos en la sección 3.2.1).

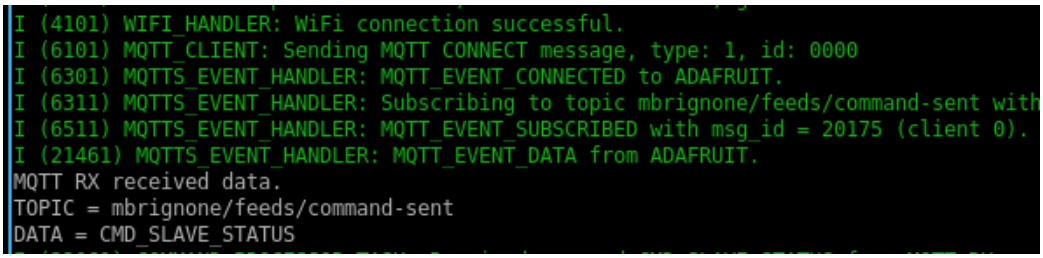

FIGURA 4.3: Eventos disparados en el microcontrolador por la comunicación utilizando MQTT.

Primero se espera a que el *driver* WiFi indique que la conexión ya está habilitada, y luego se envía la orden de conectarse al *broker* MQTT, disparándose a continuación el evento MQTT_EVENT_CONNECTED, el cual indica que la conexión al *broker* fue exitosa.

Para detectar los comandos enviados por el usuario, el cliente se suscribe al *topic* correspondiente, lo cual dispara el evento MQTT_EVENT_SUBSCRIBED.

Finalmente, se dispara el evento MQTT_EVENT_DATA indicando que hay un dato nuevo en el *topic* al cual el microcontrolador se encuentra suscrito, y se puede ver que el dato recibido se corresponde al comando enviado desde Adafruit.

Comunicación con el fabricante

La comunicación con el fabricante consiste en el envío de datos desde el microcontrolador hacia Google Cloud utilizando el protocolo MQTT. Para ello primero el dispositivo debe conectarse al *broker* de Google Cloud y autenticarse mediante un JSON Web Token, tal como se explicó en la sección 3.3. Este proceso se puede observar en la figura 4.4, donde se aprecia cómo se generó el JWT, para lo cual se obtuvo el tiempo actual mediante el protocolo SNTP, y luego se conectó a Google Cloud.

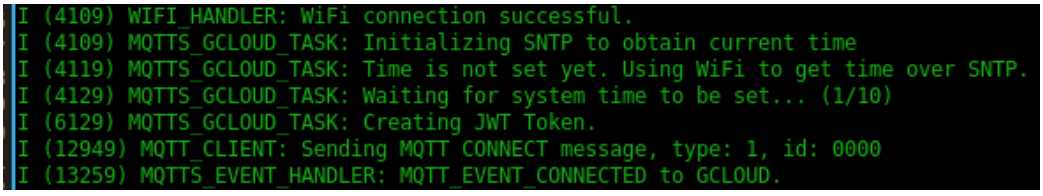

FIGURA 4.4: Conexión del microcontrolador a Google Cloud por MQTT.

Una vez conectado al *broker*, se enviaron algunos mensajes y luego se revisó el registro de eventos (*logs*) de Google Cloud para verificar que la comunicación fue exitosa. Esto se ilustra en la figura 4.5, en la que se ve que primero se produce la conexión de un dispositivo, luego la publicación de dos mensajes (con intervalos de 30 segundos entre ellos) y finalmente la desconexión del dispositivo. Cabe mencionar que en el registro se cuenta con mucha más información que la mostrada en la figura 4.5, pero se la omite por simplicidad.

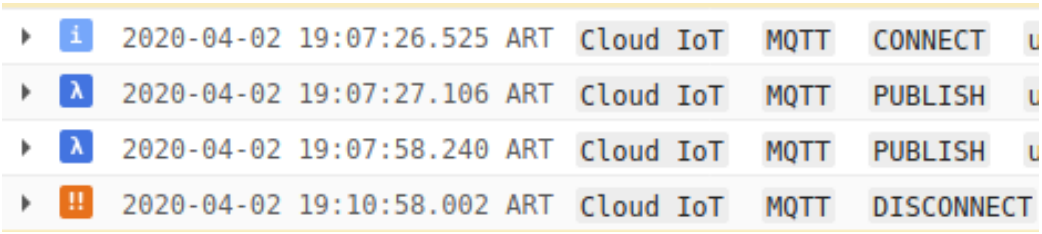

FIGURA 4.5: Registro de eventos MQTT en Google Cloud.

Servidor web

Para probar el funcionamiento del servidor web, es necesario conectarse a la red local que el propio módulo genera, cuyo nombre es ESP32_AP. Una vez conectado, es posible acceder al servidor ingresando a una dirección IP específica desde cualquier navegador.

La interfaz del servidor web es sumamente sencilla, y solamente consiste en un formulario para ingresar las credenciales de la red WiFi a la cual se desea conectar, tal como se observa en la figura 4.6.

FIGURA 4.6: Interfaz del servidor web que se ejecuta en el módulo.

Desde el punto de vista del firmware, en la figura 4.7 se observan los diferentes eventos que ocurrieron al conectarse a la red local del módulo funcionando como punto de acceso.

Primero se detectó la conexión de un nuevo cliente (*station*) al punto de acceso. Debido a que el objetivo del servidor web es permitir configurar o modificar las credenciales WiFi, se observa cómo el módulo se desconectó en ese momento de la red WiFi a la que se encontraba conectado. Luego el servidor web detectó que se envió el formulario, mostrando por consola las credenciales de la red ingresada. Finalmente el firmware desconectó automáticamente al cliente que estaba en el servidor web y procedió a conectarse a la nueva red.

```
I (34065) WIFI_HANDLER: New station connected to AP.
I (34065) WIFI_HANDLER: Disconnecting station from Access Point.
I (34065) WIFI_HANDLER: WiFi intentionally disconnected, not trying to reconnect.
I (34105) tcpip_adapter: softAP assign IP to station,IP is: 192.168.1.2
I (153225) WEB_SERVER_TASK: Received form with body:
ssid=example_name&password=123456
I (153245) WIFI_HANDLER: A station disconnected.
I (153245) WIFI_HANDLER: Reconnecting station to Access Point.
```

FIGURA 4.7: Salida por consola del microcontrolador durante la configuración de las credenciales de la red WiFi.

4.1.2. Comunicación BLE

Para probar el funcionamiento de la comunicación por Bluetooth Low Energy, se utilizó en un celular la aplicación móvil *nRF Connect*, que ofrece un poderoso conjunto de herramientas para comunicarse con dispositivos BLE. Entre ellas se encuentra la posibilidad de descubrir servicios/características y realizar operaciones de lectura/escritura sobre ellas.

Al iniciar el módulo, el WiFi se encuentra encendido por defecto, y por lo tanto el BLE está apagado, ya que ambos no pueden coexistir en simultáneo. Por ello como primera prueba se envió un comando para encender el BLE, para verificar que el WiFi se desactiva automáticamente y el proceso de BLE se inicializa correctamente. En la figura 4.8 se observa cómo se inicia el controlador Bluetooth y luego se disparan cuatro eventos asociados al servidor BLE (como se explicó en la sección 3.2.2). Estos eventos indican que se registró la aplicación, que se creó e inició un servicio, y que se agregó una característica a dicho servicio.

```
I (93449) BTDM_INIT: BT controller compile version [e736f10]
I (93449) system_api: Base MAC address is not set, read default base MAC address
I (93779) GATTS_TASK: REGISTER_APP_EVT - app_id = 0

I (93789) GATTS_TASK: CREATE_SERVICE_EVT, status 0,  service_handle 40

I (93799) GATTS_TASK: SERVICE_START_EVT, status 0, service_handle 40

I (93799) GATTS_TASK: ADD_CHAR_EVT, status 0,  attr_handle 42, service_handle 40
```

FIGURA 4.8: Inicialización de la interfaz BLE en el módulo.

Una vez inicializada la comunicación BLE, es posible descubrir el módulo utilizando la aplicación *nRF Connect*, como se observa en la figura 4.9.

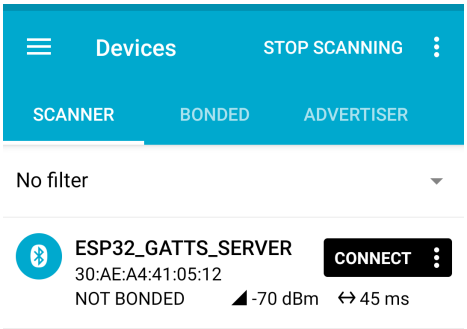

FIGURA 4.9: Servidor BLE del módulo visible en la aplicación móvil nRF Connect.

Al establecer una conexión con el módulo desde la aplicación, se puede acceder a sus servicios y características. Esto queda en evidencia en la figura 4.10, en la que se ve el servicio con su correspondiente característica, los cuales fueron previamente configurados en el firmware del módulo. Ambos figuran como *unknown* debido a que los identificadores no se corresponden con ningún valor predefinido por el *Bluetooth Special Interest Group*.

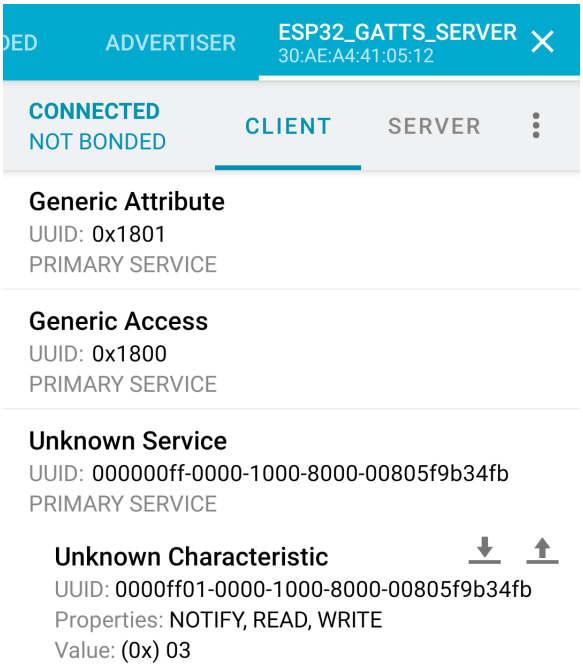

FIGURA 4.10: Servicio con su correspondiente característica, vistos en la aplicación móvil nRF Connect.

Los símbolos de flecha ubicados a la derecha de la característica permiten leer o escribir un valor, disparando así en el firmware los eventos correspondientes (figura 4.11, con tres operaciones de lectura y una de escritura con valor 0x01).

```
I (495779) GATTS_TASK: GATT_READ_EVT, conn_id 0, trans_id 8,

I (502299) GATTS_TASK: GATT_READ_EVT, conn_id 0, trans_id 9,

I (503539) GATTS_TASK: GATT_READ_EVT, conn_id 0, trans_id 10,

I (509499) GATTS_TASK: GATT_WRITE_EVT, conn_id 0, trans_id 11
I (509509) GATTS_TASK: GATT_WRITE_EVT, value len 1, value :
I (509509) GATTS_TASK: 01
```

FIGURA 4.11: Eventos disparados en el firmware ante operaciones de lectura y escritura de una característica BLE.

4.1.3. Comunicación con electrodoméstico

Para probar la comunicación serie por I2C con el electrodoméstico, emulado por otro microcontrolador ESP32, se conectaron las respectivas interfaces I2C como

se muestra en la figura 4.12. Adicionalmente se interconectaron los terminales de masa de ambos microcontroladores.

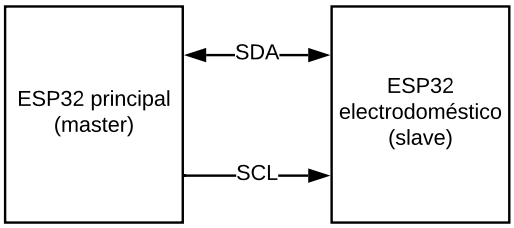

FIGURA 4.12: Conexión de las interfaces I2C de los microcontroladores.

También fue necesario ejecutar un firmware más sencillo en el microcontrolador auxiliar, con una tarea principal que periódicamente lee la información que llega por I2C y en función de ello actualiza una máquina de estados, la cual representa el estado en el que se encuentra el electrodoméstico en un momento dado.

De esta forma fue posible verificar el funcionamiento de la comunicación serie, al enviar un comando desde el módulo principal hacia el microcontrolador que emula el electrodoméstico, y comprobando que se reciben correctamente. Estas pruebas se llevaron a cabo utilizando el módulo UART para enviar comandos al bloque procesador de comandos, que se encarga de transmitir la orden al electrodoméstico (a través de la tarea encargada de la comunicación con él).

En la figura 4.13 se observa la salida por consola de las pruebas efectuadas.

```
(9663) UART_TASK: Received from UART: 1
(9663) COMMAND_PROCESSOR_TASK: Received command CMD_SLAVE_START_B from UART_RX.
(10523) I2C_MASTER_TASK: Received 1 from Command Processor, sending it to slave.
(10623) COMMAND_PROCESSOR_TASK: Command successfully sent to Slave.
(23283) UART_TASK: Received from UART: 5
(23283) COMMAND_PROCESSOR_TASK: Received command CMD_SLAVE_STATUS from UART_RX.
(23623) I2C_MASTER_TASK: Received 5 from Command Processor, sending it to slave.
(23723) COMMAND_PROCESSOR_TASK: Command successfully sent to Slave.
(23723) I2C_MASTER_TASK: Reading Slave status.
(23723) COMMAND_PROCESSOR_TASK: Slave FSM current state: 3
(23843) UART_TASK: Received from Command Processor: 3
```

FIGURA 4.13: Salida por consola del módulo al comunicarse con el electrodoméstico.

Primero se envió por UART el comando 1, que llegó al procesador de comandos y se detectó que se corresponde con una acción a disparar en el electrodoméstico, por lo que se lo envió al módulo encargado de la comunicación serie (I2C_MASTER_TASK). Este último envió el comando al otro microcontrolador (el esclavo o *slave* I2C), indicándole al procesador de comandos que pudo hacerlo exitosamente.

Luego se envió por UART el comando 5, el cual indica que se desea conocer el estado del electrodoméstico. La primera parte es idéntica al caso del comando anterior, pero posteriormente la tarea de comunicación serie lee el estado del otro microcontrolador y se lo comunica al procesador de comandos. En este caso se recibió un 3, que se corresponde con el estado SLAVE_PROCESS_B.

En la figura 4.14 se aprecia lo que ocurrió desde el punto de vista del microcontrolador esclavo que emula al electrodoméstico, pudiéndose ver que el comportamiento es consistente con las órdenes enviadas por el módulo principal. Primero se recibió el comando 1 y la máquina de estados pasó al estado SLAVE_PROCESS_B. Luego se recibió el comando 5 y se envió por I2C el estado actual en el que se encontraba.

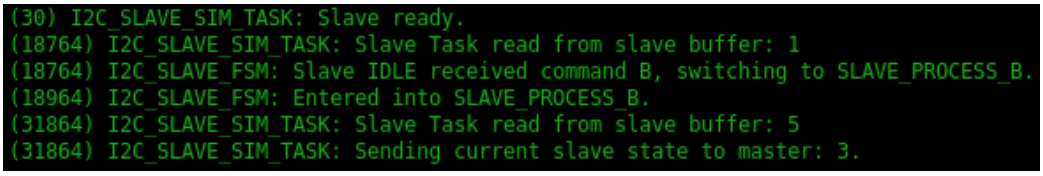

FIGURA 4.14: Salida por consola del microcontrolador que emula al electrodoméstico, al recibir comandos desde el microcontrolador principal.

4.2. Integración del sistema

Para probar el funcionamiento general del sistema, todas las tareas asociadas a los diferentes módulos de firmware deben estar ejecutándose en simultáneo. Esto permite verificar el flujo completo, desde el envío de un comando por parte del usuario (usando la plataforma web) hasta la ejecución de la acción correspondiente en el electrodoméstico (emulado por otro microcontrolador).

En la figura 4.15 se observa cómo se inicializan las diferentes conexiones del sistema. Primero se produce la conexión a la red WiFi y comienza a ejecutarse el servidor web. Una vez conectado a la red, se producen las conexiones con los *brokers* MQTT: Adafruit (para la interacción con el usuario) y Google Cloud (para el envío de datos al fabricante), que requiere también la generación del JSON Web Token (JWT).

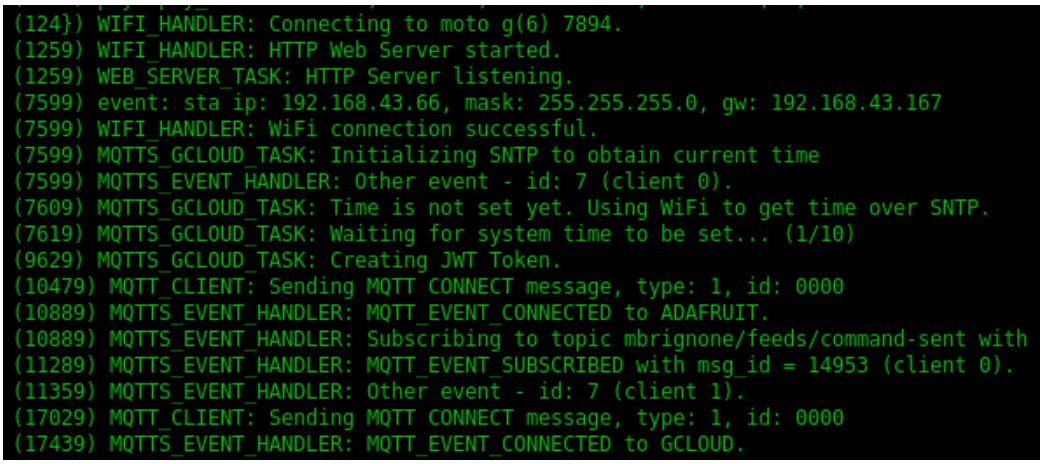

FIGURA 4.15: Salida por consola al inicializar todos los módulos del sistema.

A continuación (figura 4.16) se puede observar cómo se envía periódicamente a Google Cloud la información de estado del electrodoméstico. Para ello, la tarea de Google Cloud le envía el comando CMD_SLAVE_STATUS al procesador de

comandos para obtener el estado del electrodoméstico, de la misma forma que lo haría el usuario. El procesador de comandos obtiene el estado de la forma explicada en la sección 4.1.3 y se lo pasa de regreso a la tarea de Google Cloud, que se encarga de publicarlo. Se puede observar también el formato JSON que se utiliza para enviar la información de interés.

```
(13590) COMMAND_PROCESSOR_TASK: Received command CMD_SLAVE_STATUS from MQTT_GCLOUD.
(13600) I2C_MASTER_TASK: Received 5 from Command Processor, sending it to slave.
(13700) I2C_SLAVE_SIM_TASK: Sending current slave state to master: 0.
(13710) COMMAND_PROCESSOR_TASK: Command successfully sent to Slave.
(13710) I2C_MASTER_TASK: Reading Slave status.
(13710) COMMAND_PROCESSOR_TASK: Slave FSM current state: 0
(13710) MQTTS_GCLOUD_TASK: Received from Command Processor TASK: SLAVE_IDLE (0)
(13720) MQTTS_GCLOUD_TASK: Publishing to Google Cloud.
(13730) MQTTS_GCLOUD_TASK: JSON value = {"timestamp": 1586099223, "device": "esp32_real", "state":
'SLAVE_IDLE", "state_int": 0, "temp": 25}
(13740) MQTTS_GCLOUD_TASK: Sent publish successful.
```

FIGURA 4.16: Salida por consola al momento de enviar datos a Google Cloud.

La figura 4.17 muestra la salida por consola cuando el usuario envía un comando desde la plataforma web, evidenciando un comportamiento muy similar al caso anterior.

```
I (23510) MQTTS_EVENT_HANDLER: MQTT_EVENT_DATA from ADAFRUIT.
MQTT RX received data.
TOPIC = mbrignone/feeds/command-sent
DATA = CMD_SLAVE_STATUS
I (23700) COMMAND_PROCESSOR_TASK: Received command CMD_SLAVE_STATUS from MQTT_RX.
I (23750) I2C_MASTER_TASK: Received 5 from Command Processor, sending it to slave.
I (23850) COMMAND_PROCESSOR_TASK: Command successfully sent to Slave.
I (23850) I2C_MASTER_TASK: Reading Slave status.
I (23850) COMMAND_PROCESSOR_TASK: Slave FSM current state: 3
I (24270) MQTTS_USER_TASK: Received from Command Processor TASK: SLAVE_PROCESS_B (3)
I (25270) MQTTS_USER_TASK: Sent publish successful.
```

FIGURA 4.17: Salida por consola al momento de recibir un comando del usuario por WiFi.

Una prueba importante al integrar el sistema consiste en verificar la correcta transición entre la conexión WiFi y la conexión BLE. Esto requiere especial cuidado ya que al activar una, automáticamente se desactiva la otra, deshabilitando también todas las conexiones que hubiese en ese momento. Posteriormente, al volver a activarse la primera, se deben restablecer todas las conexiones satisfactoriamente.

Al iniciar el sistema, la conexión WiFi se encuentra habilitada, por lo que desde la plataforma WiFi se envía el comando CMD_BLE para habilitar el BLE, desencadenando los eventos de la figura 4.18.

Al recibir el comando, se desactiva la conexión WiFi y se inicializa el servidor BLE, disparando los eventos correspondientes para crear e iniciar el servicio.

El *event handler* del *driver* WiFi detecta que se desconectó intencionalmente de la red, por lo que no intenta volver conectarse automáticamente. El WiFi se deshabilita de manera asincrónica, por lo que se interrumpen todas las conexiones que estaban activas y por ello aparecen errores de conexión en la salida por consola (en rojo y etiquetados con la letra E en la figura 4.18).

```
I (35091) MQTTS_EVENT_HANDLER: MQTT_EVENT_DATA from ADAFRUIT.
MQTT RX received data.
TOPIC = mbrignone/feeds/command-sent
DATA = CMD_BLE
I (35751) COMMAND_PROCESSOR_TASK: Received command CMD_BLE from MQTT_RX.
I (35751) COMMAND_PROCESSOR_TASK: Stopping WiFi and starting BLE server.
E (35761) event: system_event_sta_disconnected_handle_default 294 esp_wifi_internal_reg_rxcb
E (35761) esp-tls: read error :-76:
I (35771) WIFI_HANDLER: WiFi intentionally disconnected, not trying to reconnect.
E (35771) TRANS_SSL: esp_tls_conn_read error, errno=Software caused connection abort
E (35781) event: system_event_ap_stop_handle_default 223 esp_wifi_internal_reg_rxcb ret=0x30
E (35791) MQTT_CLIENT: Read error or end of stream
I (35801) MQTTS_EVENT_HANDLER: MQTT_EVENT_DISCONNECTED (client 0).
I (35811) BTDM_INIT: BT controller compile version [e736f10]
I (35821) system_api: Base MAC address is not set, read default base MAC address from BLK0 o
I (36161) GATTS_TASK: REGISTER_APP_EVT - app_id = 0
I (36171) GATTS_TASK: CREATE_SERVICE_EVT, status 0,  service_handle 40

I (36181) GATTS_TASK: SERVICE_START_EVT, status 0, service_handle 40
```

FIGURA 4.18: Salida por consola al habilitar la conectividad BLE, estando previamente conectado por WiFi.

Se debe notar que mientras se encuentra habilitada la conectividad BLE, y por lo tanto deshabilitado el WiFi, no es posible enviar información de estado a Google Cloud.

En este punto se puede verificar la transición en el sentido inverso, es decir pasando de BLE a WiFi. Para ello se manda el comando CMD_WIFI por Bluetooth, lo cual desactiva el BLE y habilita el WiFi, desencadenando los eventos de la figura 4.19. Se puede observar cómo se produce otra vez la inicialización de la conexión WiFi y del servidor web, para luego proceder a realizar exitosamente la conexión por MQTT a ambos *brokers* (Adafruit para el usuario y Google Cloud para el fabricante). Nuevamente hay errores de conexión en el medio debido a la transición, pero luego se conecta correctamente.

```
(287280) COMMAND_PROCESSOR_TASK: Stopping BLE server and starting WiFi.
(287290) BT_APPL: bta_dm_disable BTA_DISABLE_DELAY set to 200 ms
(287500) WIFI_HANDLER: DHCP server stopped.
(287510) WIFI_HANDLER: TCP adapter configured with IP 192.168.1.1/24 .
(287510) WIFI_HANDLER: DHCP server started.
(287550) system_api: Base MAC address is not set, read default base MAC address from BLK0
(287550) system_api: Base MAC address is not set, read default base MAC address from BLK0
(288580) WIFI_HANDLER: Connecting to Family B Ext.
(288580) MQTTS_EVENT_HANDLER: Other event - id: 7 (client 0).
(288580) WIFI_HANDLER: HTTP Web Server started.
(288580) WEB_SERVER_TASK: HTTP Server listening.
(291640) event: sta ip: 192.168.0.4, mask: 255.255.255.0, gw: 192.168.0.1
(291640) WIFI_HANDLER: WiFi connection successful.
(297990) MQTTS_EVENT_HANDLER: Other event - id: 7 (client 1).
(303410) MQTT_CLIENT: Sending MQTT CONNECT message, type: 1, id: 0000
(303710) MQTTS_EVENT_HANDLER: MQTT_EVENT_CONNECTED to GCLOUD.
(306830) esp-tls: Failed to connnect to host (errno 113)
(306830) esp-tls: Failed to open new connection
(306830) TRANS_SSL: Failed to open a new connection
(306830) MQTT_CLIENT: Error transport connect
(306840) MQTTS_EVENT_HANDLER: MQTT_EVENT_DISCONNECTED (client 0).
(326840) MQTTS_EVENT_HANDLER: Other event - id: 7 (client 0).
(329110) MQTT_CLIENT: Sending MQTT CONNECT message, type: 1, id: 0000
(329310) MQTTS_EVENT_HANDLER: MQTT_EVENT_CONNECTED to ADAFRUIT.
```

FIGURA 4.19: Salida por consola al habilitar la conectividad WiFi, estando previamente conectado por BLE.

4.2.1. Visualización de datos en Google Cloud Platform

El último aspecto que resta mostrar es el procesamiento y visualización de datos en Google Cloud Platform. La información enviada por el dispositivo a Google Cloud se puede acceder desde múltiples servicios dentro de la plataforma.

Por un lado, como ya se mostró en la figura 4.5, es posible ver los *logs* y conocer con precisión en qué momento se produjeron las conexiones, envío de datos y desconexiones de los distintos dispositivos. Además, en ese punto se puede filtrar con facilidad para obtener solamente los errores que hayan ocurrido en la comunicación.

Por otro lado, toda la información publicada se puede ver en el *topic* correspondiente creado en el servicio de Cloud Pub/Sub. Para ello se debe crear una suscripción al tema en cuestión y hacer una solicitud de los datos, obteniendo como resultado lo que se observa en la figura 4.20.

Publish time	Attribute keys	Message body ↑	Body JSON keys	attribute.deviceId	attribu
Apr 4, 2020, 9:47:41 PM	deviceId	{"timestamp": 1586...	timestamp	esp32_real	26820
Apr 4, 2020, 9:48:12 PM	deviceId	{"timestamp": 1586...	timestamp	esp32_real	26820
Apr 4, 2020, 9:48:43 PM	deviceId	{"timestamp": 1586...	timestamp	esp32_real	26820

FIGURA 4.20: Datos publicados en un *topic* de Cloud Pub/Sub por el microcontrolador.

Sin embargo, la mejor forma de analizar y procesar los datos crudos es a través de la base de datos en el servicio Big Query. Gracias a la arquitectura de la figura 3.7, todos los datos publicados en Cloud Pub/Sub son almacenados en Big Query, mediante el servicio Cloud Dataflow, que también se encarga de transformar los datos publicados con formato JSON en valores dentro de columnas de una tabla en la base de datos. De esta forma, en Big Query se obtiene la información como se muestra en la figura 4.21, en la que se ve claramente la hora a la que se registró el dato, a qué dispositivo corresponde, el estado en el que estaba (representado como cadena de caracteres y como número) y la temperatura a la que se encontraba.

Row	timestamp	device	state	state_int	temp
1	2020-04-04 23:57:45 UTC	esp32_real	SLAVE_PROCESS_B	3	27
2	2020-04-04 23:57:14 UTC	esp32_real	SLAVE_IDLE	0	28
3	2020-04-04 23:56:43 UTC	esp32_real	SLAVE_IDLE	0	26
4	2020-04-04 23:56:12 UTC	esp32_real	SLAVE_IDLE	0	27
5	2020-04-04 23:55:41 UTC	esp32_real	SLAVE_IDLE	0	25
6	2020-04-04 23:55:10 UTC	esp32_real	SLAVE_IDLE	0	26
7	2020-04-04 23:54:39 UTC	esp32_real	SLAVE_IDLE	0	24

FIGURA 4.21: Datos obtenidos de la base de datos en Big Query.

Al estar la información almacenada en una base de datos, es posible obtener datos de casos específicos mediante consultas SQL (*Structured Query Language*) [40], lo que brinda una herramienta de análisis sumamente poderosa. Por ejemplo, con las sentencias del algoritmo 4.1 se pueden obtener todos los registros en los que el dispositivo con ID «esp32_real» tuvo una temperatura mayor a 40 grados. Los resultados que se obtienen con la consulta se pueden observar en la figura 4.22.

```
1 SELECT * FROM esp32_dataset.esp32_data
2 WHERE device="esp32_real" AND temp > 40
3 ORDER BY timestamp DESC
```

ALGORITMO 4.1: Consulta SQL para obtener información de la base de datos en Big Query.

Row	timestamp	device	state	state_int	temp
1	2020-04-05 00:12:47 UTC	esp32_real	SLAVE_PROCESS_B	3	43
2	2020-04-05 00:12:16 UTC	esp32_real	SLAVE_PROCESS_B	3	44
3	2020-04-05 00:11:45 UTC	esp32_real	SLAVE_PROCESS_B	3	42
4	2020-04-05 00:11:14 UTC	esp32_real	SLAVE_PROCESS_B	3	43
5	2020-04-05 00:10:43 UTC	esp32_real	SLAVE_PROCESS_B	3	41

FIGURA 4.22: Datos obtenidos de la base de datos con la consulta del algoritmo 4.1.

Además de acceder a los datos crudos, es posible obtener diferentes gráficos para analizar con mayor facilidad la información. En la figura 4.23 se puede observar el más simple de ellos, en el que se muestran los dispositivos activos a lo largo del tiempo. En este caso se utilizó un único módulo para enviar mensajes a Google Cloud, por lo tanto el número máximo de dispositivos es siempre 1. Este gráfico fue elaborado utilizando el servicio de *Monitoring*, ofrecido también por la propia plataforma, y que permite monitorear prácticamente cualquier recurso que se utilice.

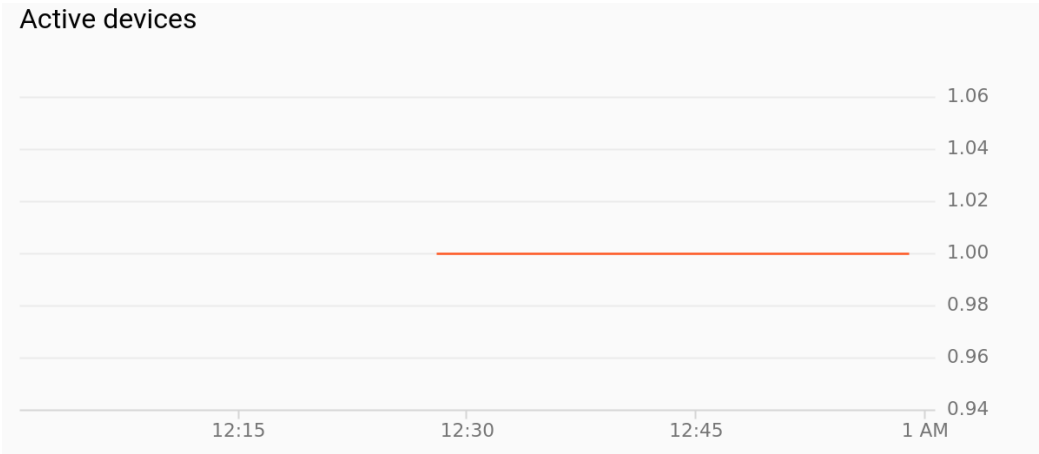

FIGURA 4.23: Gráfico con los dispositivos activos a lo largo del tiempo.

Las figuras 4.24 y 4.25 muestran otros gráficos que el servicio de *Monitoring* permite efectuar. En el primer caso se grafica la cantidad total de operaciones de publicación de mensajes a lo largo del tiempo. En el segundo caso se muestra la cantidad total de bytes de información enviados desde los dispositivos hacia Google Cloud, lo cual permite tener una noción exacta del ancho de banda que se está utilizando.

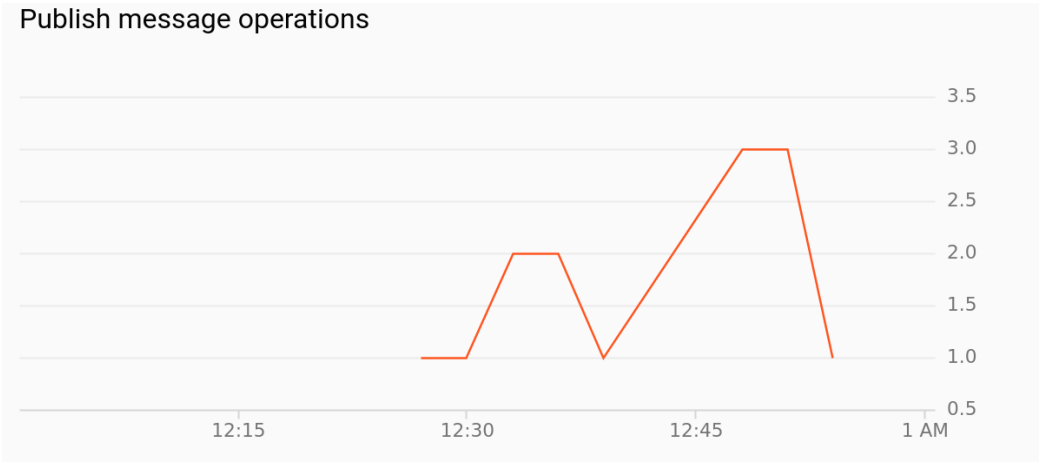

FIGURA 4.24: Gráfico con la cantidad de operaciones de publicación de mensajes a lo largo del tiempo.

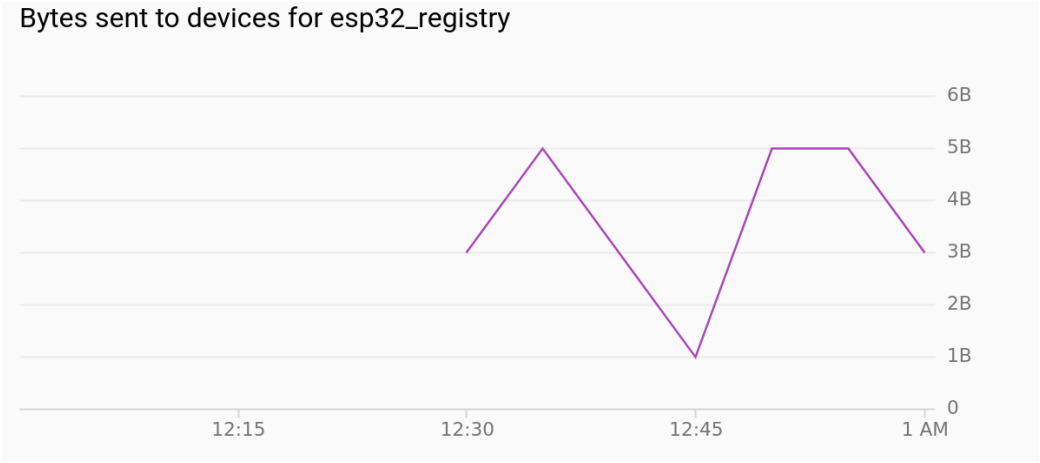

FIGURA 4.25: Gráfico con la cantidad de bytes de información enviados a Google Cloud.

Finalmente, para completar el flujo de la arquitectura implementada en Google Cloud Platform, la información almacenada en Big Query se puede utilizar como fuente de datos para Grafana, permitiendo confeccionar gráficos personalizados para visualizar las variables de interés. La versatilidad en los gráficos que se pueden elaborar es muy amplia, ya que para determinar los datos a mostrar se utilizan consultas SQL que permiten filtrar entradas de la base de datos con facilidad.

En la figura 4.26 se muestra un gráfico que se elaboró utilizando Grafana, en el que se grafica la temperatura reportada por el microcontrolador a lo largo del tiempo. Se hace una distinción de los valores reportados de acuerdo al estado del electrodoméstico en ese momento (color verde para las temperaturas estando en

SLAVE_IDLE y color naranja para el estado SLAVE_PROCESS_B), lo cual posibilitaría encontrar una relación entre la temperatura y el estado del dispositivo. También se muestran los valores mínimo, máximo, promedio y actual (el último medido) para cada estado. Además, en el gráfico se traza un limite en una temperatura de 40 grados (la zona roja del gráfico), para poder visualizar fácilmente los puntos que superen ese umbral.

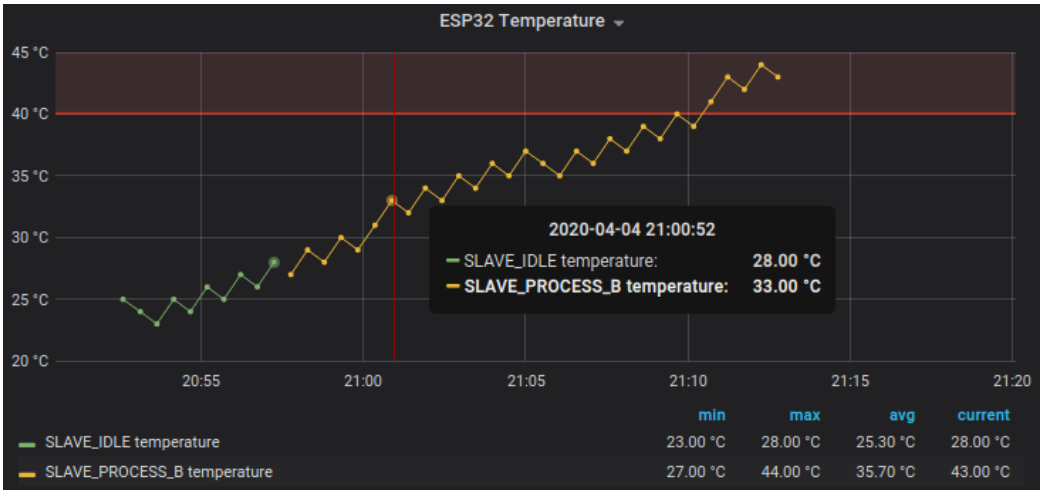

FIGURA 4.26: Gráfico elaborado con Grafana a partir de los datos almacenados en la base de datos de Big Query.

El verdadero potencial de Google Cloud se puede aprovechar cuando se tienen múltiples dispositivos conectados enviando datos, pero en este caso solamente se cuenta con un único módulo de hardware para publicar información. Para solventar esta problemática, se utilizó un *script* en Python para simular dispositivos adicionales conectados, publicando mensajes periódicamente de la misma forma que lo haría un electrodoméstico real. Así se obtiene el gráfico de la figura 4.27, en el que se muestran múltiples dispositivos en simultáneo a lo largo del tiempo.

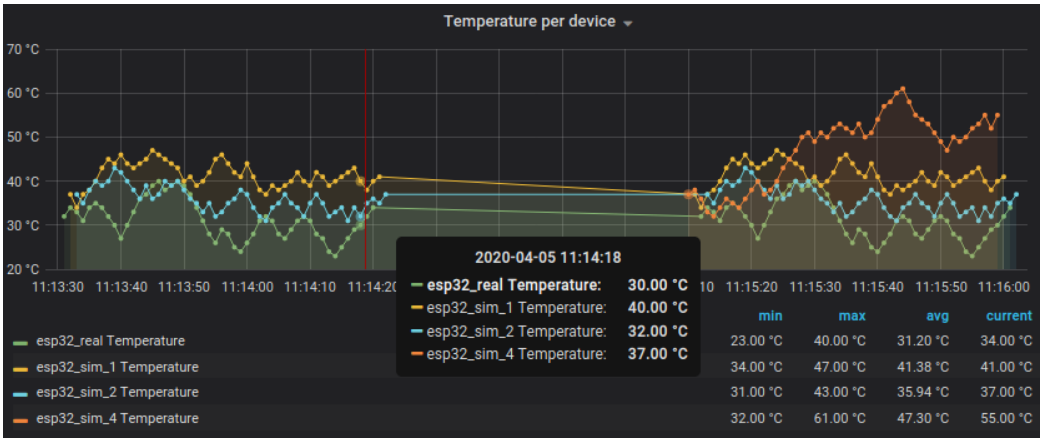

FIGURA 4.27: Gráfico elaborado con Grafana a partir de los datos en Big Query de varios dispositivos.

Capítulo 5

Conclusiones

En este capítulo se mencionan los aspectos más importantes del trabajo realizado y se contemplan los próximos pasos a seguir en el desarrollo.

5.1. Conclusiones generales

En este trabajo se logró implementar de manera exitosa un módulo capaz de dotar de conectividad WiFi y Bluetooth a un electrodoméstico. Para ello se diseñaron e implementaron diferentes módulos de firmware que permiten comunicarse con el usuario mediante la recepción de comandos por WiFi/Bluetooth, los cuales luego son retransmitidos al electrodoméstico (emulado por otro microcontrolador). También permiten que el fabricante cuente con información de estado en tiempo real de sus electrodomésticos, a través del envío y procesamiento de datos en Google Cloud Platform.

Durante el desarrollo del trabajo, fueron indispensables los conocimientos adquiridos en las diferentes materias de la Carrera de Especialización en Sistemas Embebidos, entre las que se destacan las siguientes:

- Sistemas operativos de tiempo real I y II: constituyeron el primer contacto con un RTOS y permitieron aprender acerca del uso de tareas, recursos de sincronización y comunicación entre ellas, y uso de memoria en el contexto de un RTOS.
- Protocolos de comunicación en sistemas embebidos: la interfaces de comunicación vistas (WiFi, Bluetooth, I2C) fueron una parte integral del trabajo realizado, al igual que otros protocolos de nivel superior (como HTTP para la capa de aplicación).
- Ingeniería de software en sistemas embebidos: permitió adquirir buenas prácticas de desarrollo de software, y elaborar el diseño de algunos módulos de firmware en una etapa temprana del trabajo.

5.2. Próximos pasos

El próximo paso lógico para el trabajo realizado consiste en la integración del módulo a un electrodoméstico real, para demostrar así el valor que aporta en un entorno más realista. Esto implicaría también el diseño y fabricación de un circuito impreso que se adapte al electrodoméstico seleccionado.

Además, existen una serie de funcionalidades adicionales que sería deseable implementar a futuro, a los fines de contar con un desarrollo aún más completo. Algunas de estas funcionalidades se presentan a continuación:

- Agregar soporte para más interfaces de comunicación con el electrodoméstico: las pruebas se realizaron sólo con I2C, por lo que sería deseable agregar al menos SPI y UART, e incluso evaluar otros protocolos adicionales.
- Mejorar servidor web: algunas funcionalidades que se le podrían incorporar son la capacidad de mostrar las redes WiFi disponibles y el uso de encriptación para proteger la información que se intercambia al configurar las credenciales.
- Realizar actualizaciones remotas: conocidas como actualizaciones OTA (*over the air*), permitirían que el fabricante actualice el firmware de todos sus electrodomésticos conectados de manera remota. El *framework* del microcontrolador utilizado tiene soporte para recibir estas actualizaciones, y Google Cloud permite enviarlas, por lo que sería factible implementarlo.

Bibliografía

[1] IHS Markit. *The Internet of Things: a movement, not a market*. 2017.

[2] Statista. *Forecast end-user spending on IoT solutions worldwide from 2017 to 2025*. https://www.statista.com/statistics/976313/global-iot-market-size/. 2017. (Visitado 14-03-2020).

[3] Cloudflare. *What is the Mirai Botnet?* https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/. (Visitado 15-03-2020).

[4] Zack Whittaker. *CIA, MI5 hacked smart TVs to eavesdrop on private conversations*. https://www.zdnet.com/article/how-cia-mi5-hacked-your-smart-tv-to-spy-on-you/. 2017. (Visitado 15-03-2020).

[5] Danny Palmer. *Ransomware, snooping and attempted shutdowns: see what hackers did to these systems left unprotected online*. https://www.zdnet.com/article/ransomware-snooping-and-attempted-shutdowns-the-state-of-this-honeypot-shows-what-hackers-do-to-systems-left-unprotected-online/. 2020. (Visitado 15-03-2020).

[6] https://thingsboard.io/. (Visitado 18-03-2020).

[7] https://thinger.io/. (Visitado 18-03-2020).

[8] https://ubidots.com/. (Visitado 18-03-2020).

[9] *ThingWorx*. https://www.ptc.com/-/media/Files/PDFs/IoT/ThingWorx-Connect-Product-Brief.pdf. (Visitado 18-03-2020).

[10] https://www.particle.io/. (Visitado 18-03-2020).

[11] Asociación Española de Domótica e Inmótica. *Qué es Domótica*. http://www.cedom.es/sobre-domotica/que-es-domotica. (Visitado 17-03-2020).

[12] *NB-IoT, la nueva revolución del mundo conectado*. https://accent-systems.com/es/nb-iot. (Visitado 24-03-2020).

[13] Semtech. *What is Lora?* https://www.semtech.com/lora/what-is-lora. (Visitado 24-03-2020).

[14] Sigfox. *Sigfox technology*. https://www.sigfox.com/en/what-sigfox/technology. (Visitado 24-03-2020).

[15] *Wifi*. https://es.wikipedia.org/wiki/Wifi. (Visitado 24-03-2020).

[16] *Learn about Bluetooth Technology*. https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/. (Visitado 24-03-2020).

[17] *TCP/IP*. https://es.ccm.net/contents/282-tcp-ip-que-significa-tcp-ip. (Visitado 25-03-2020).

[18] *Generalidades del protocolo HTTP*. https://developer.mozilla.org/es/docs/Web/HTTP/Overview. (Visitado 25-03-2020).

[19] *HTTPS*. https://es.ryte.com/wiki/HTTPS. (Visitado 25-03-2020).

[20] *¿Qué es MQTT? Su importancia como protocolo IoT*. https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/. (Visitado 25-03-2020).
[21] *YouTrack*. https://www.jetbrains.com/es-es/youtrack/. (Visitado 22-03-2020).
[22] *ESP32, a different IoT power and performance*. https://www.espressif.com/en/products/hardware/esp32/overview. (Visitado 26-03-2020).
[23] *NodeMCU ESP32*. https://www.infootec.net/nodemcu-esp32/. (Visitado 28-03-2020).
[24] *ESP32 WROOM Series*. https://www.espressif.com/en/products/hardware/esp-wroom-32/overview. (Visitado 28-03-2020).
[25] *FreeRTOS, Real-time operating system for microcontrollers*. https://www.freertos.org/. (Visitado 28-03-2020).
[26] https://cloud.google.com.
[27] https://aws.amazon.com/.
[28] https://azure.microsoft.com/es-es/.
[29] *Cloud IoT Core*. https://cloud.google.com/iot-core. (Visitado 29-03-2020).
[30] *FreeRTOS Queues*. https://www.freertos.org/Embedded-RTOS-Queues.html. (Visitado 29-03-2020).
[31] *Event Bits (or flags) and Event Groups*. https://www.freertos.org/FreeRTOS-Event-Groups.html. (Visitado 29-03-2020).
[32] *lwIP - A Lightweight TCP/IP stack*. http://savannah.nongnu.org/projects/lwip/. (Visitado 29-03-2020).
[33] *Seguridad de la capa de transporte*. https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte. (Visitado 29-03-2020).
[34] https://tls.mbed.org/. (Visitado 29-03-2020).
[35] *What happens in a TLS handshake?* https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/. (Visitado 30-03-2020).
[36] *Device security*. https://cloud.google.com/iot/docs/concepts/device-securityprovisioning_credentials. (Visitado 31-03-2020).
[37] *Introduction to JSON Web Tokens*. https://jwt.io/introduction/. (Visitado 31-03-2020).
[38] https://thingspeak.com/. (Visitado 01-04-2020).
[39] https://io.adafruit.com/. (Visitado 01-04-2020).
[40] *Conceptos básicos de SQL*. https://geotalleres.readthedocs.io/es/latest/conceptos-sql/conceptos_sql.html. (Visitado 04-04-2020).